Tampereen yliopisto
Tampere University

COMP.SE.100-EN (ItSE)
Introduction to Software Engneering

Any questions at Padlet ?

Tensu: remember to start Zoom lecture recording, at 1415

Prefer course Moodle over SISU information.

Students are recommended to follow Moodle News/messages.

Tampereen yliopisto
Tampere University

Course contents (plan)

1. Course basics, intro
2. **Sw Eng in general, overview**
3. Requirements
4. Different software systems
5. Basic UML Diagrams (”Class”, Use Case, Navigation)
6. Life Cycle models
7. UML diagrams, in more detail
8. Quality and Testing
9. Project work
10. Project management
11. Open source, APIs, IPR
12. Embedded systems
13. Recap

Tampereen yliopisto
Tampere University

2. Sw Eng in general, overview

- life cycles, S(D)LC
- software engineering vs. ”other” work…
- sw eng IS teamwork

- feasibility study / preliminary analysis
- stakeholder analysis

- how to get sw; buy (build), tailor/modify, COTS, SaaS,…
- it MAY be worth thinking to modify your way of work, not the software…
- ethics (Code of Conduct)
- documentation
- reuse.

Tampereen yliopisto
Tampere University

First, general course matters

- 24 full (4/4) project groups, and two groups of three students

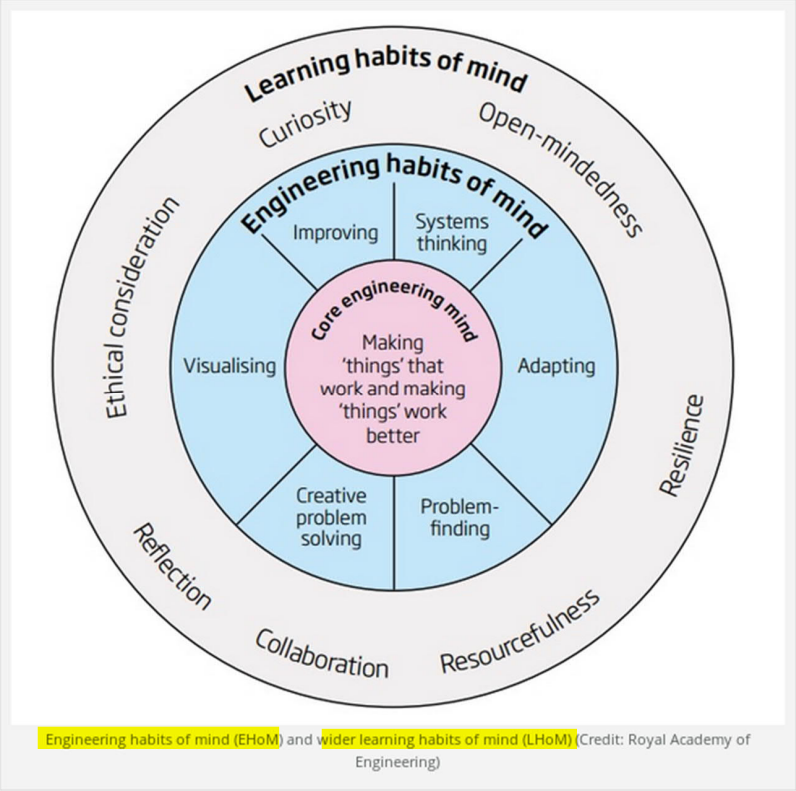
Juanita: groups G01-G04
Aleksius: ODD groups; G05,G07,G09,G11,G13,G15,G17,G19,G21,G23,G27
Lauri: EVEN groups; G06,G08,G10,G12,G14,G16,G18,G20,G22,G24,G28

- Trello board and Product Backlog ready at the end of this week (w37)
- invite your assistant to your Trello board.

- WE3 next week; feasibity study, stakeholders

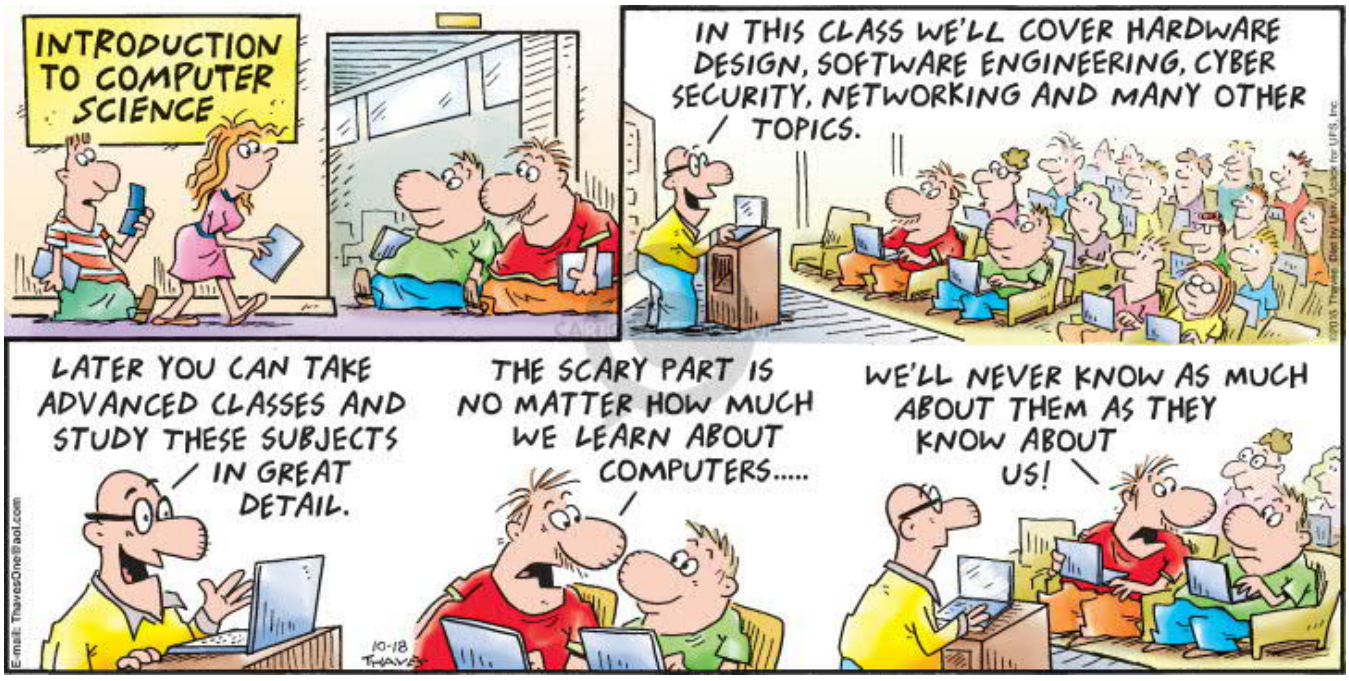
One purpose on basic sw eng courses is to teach ”engineering mind” and ”technical thinking”.

As is with mathematics courses, which teach analytical thinking and problem solving.



Engineering habits of mind (EHoM) and wider learning habits of mind (LHoM) (Credit: Royal Academy of Engineering)

Watch YouTube video; *”amazing mind reader reveals his gift”*



Software Engineering in general

Software Engineering (FI: ohjelmistotuotanto)

Software (FI: ohjelmisto)

1. computer programs, procedures and possibly associated documentation and data pertaining to the operation of a computer system.
2. all or part of the programs, procedures, rules, and associated documentation of an information processing system.
3. program or set of programs used to run a computer.

Software Engineering (FI: ohjelmistotuotanto)

1. systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software.
2. application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

[IEEE 24765-2017]

Software systems (FI: ~ ohjelmistotekniikka, terminologia vaihtelee).

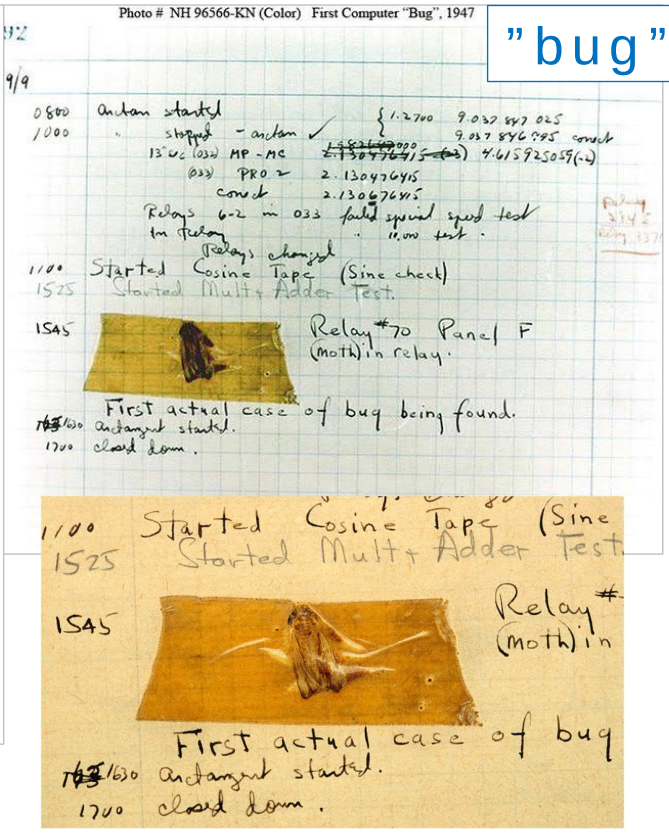
Always ready to pioneer new methods of problem-solving, Captain Grace M. Hopper made history on September 9, 1947, by removing the first computer bug. It was a moth that got into the relay calculator and was the first actual case of a bug being found.

Afterward, the operators coined the phrase that they have "debugged the computer." Of course anyone with a computer science background probably knows that story.

Captain Grace Hopper

At 3:45 p.m., Grace Murray Hopper records the first computer bug' in the Harvard Mark II computer's log book. The problem was traced to a moth stuck between relay contacts in the computer, which Hopper duly taped into the Mark II's log book with the explanation: "First actual case of bug being found." The bug was actually found by others but Hopper made the logbook entry.

[https://stationhypo.com/2019/09/09/grace-hopper-and-the-first-computer-bug/]



"bug"

Tampereen yliopisto
Tampere University

Software Engineering (Development) Life Cycle

SLC = software life cycle

tuong tuong

software life cycle. The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase. *Note:* These phases may overlap or be performed iteratively. *Contrast with:* **software development cycle.**

COMP.SE.100
ItSE is about
that small part

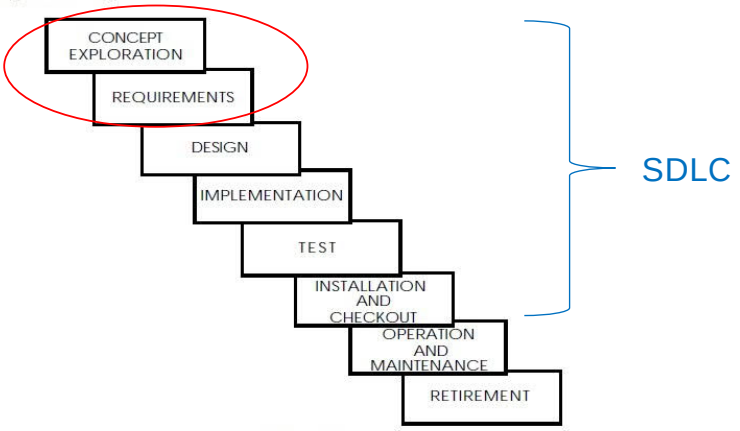


Fig 15
Sample Software Life Cycle

[IEEE Standard Glossary of Software Engineering Terminology, Std 610.12-1990(R2002)]

ISO/IEC/IEEE 15288:2015
Systems and software engineering – system life cycle processes

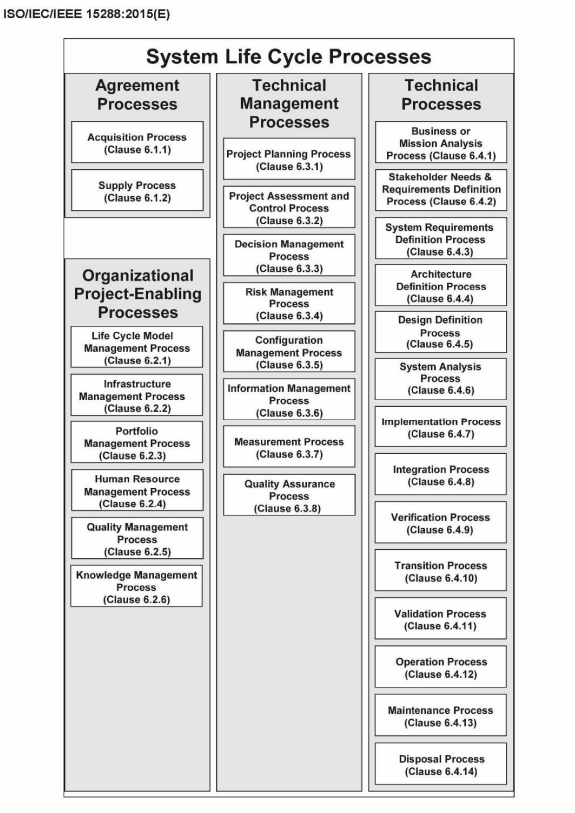


Figure 4 — System life cycle processes

stakeholder needs & requirement def process

system requirement def process

ISO/IEC/IEEE 15288:2015
Systems and software engineering – system life cycle processes

COMP.SE.100 matters are just a small part at the beginning of a life cycle.

SRS = Software Requirements Specification

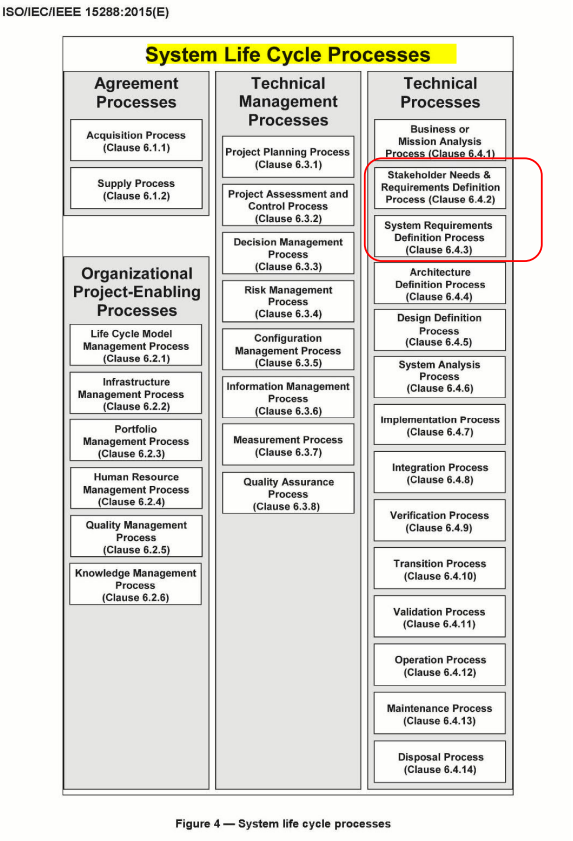
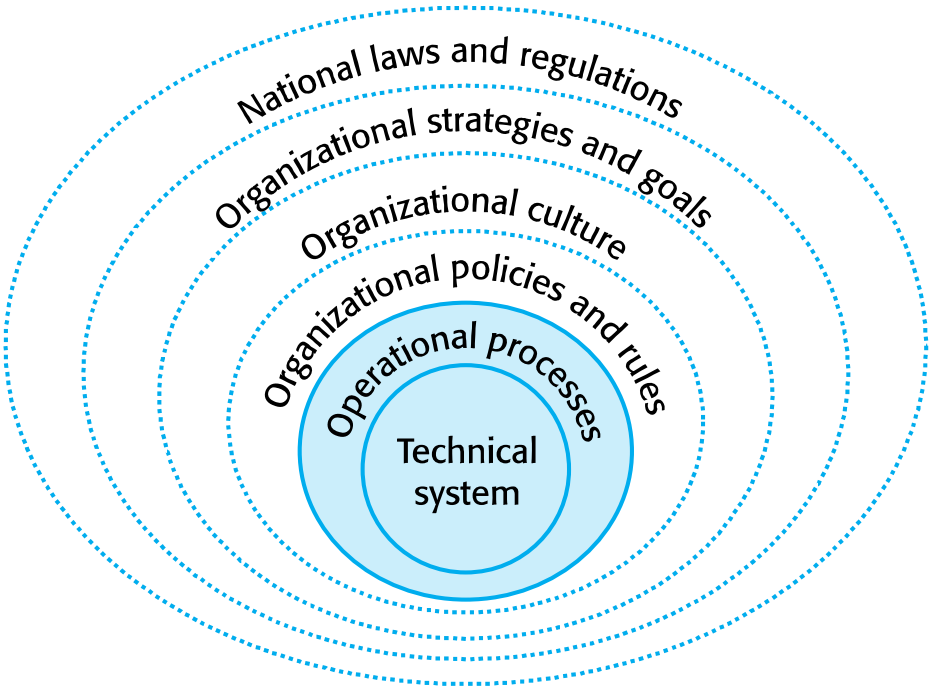
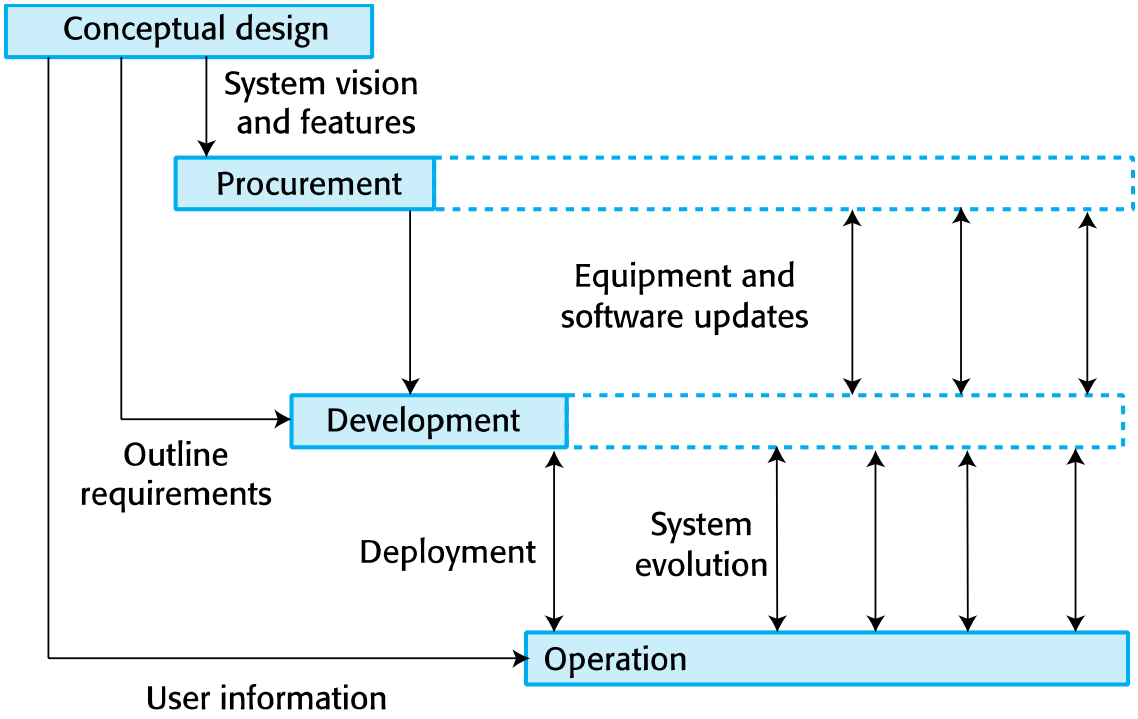


Figure 4 — System life cycle processes

System in general [Sommerville SE10, 2015]

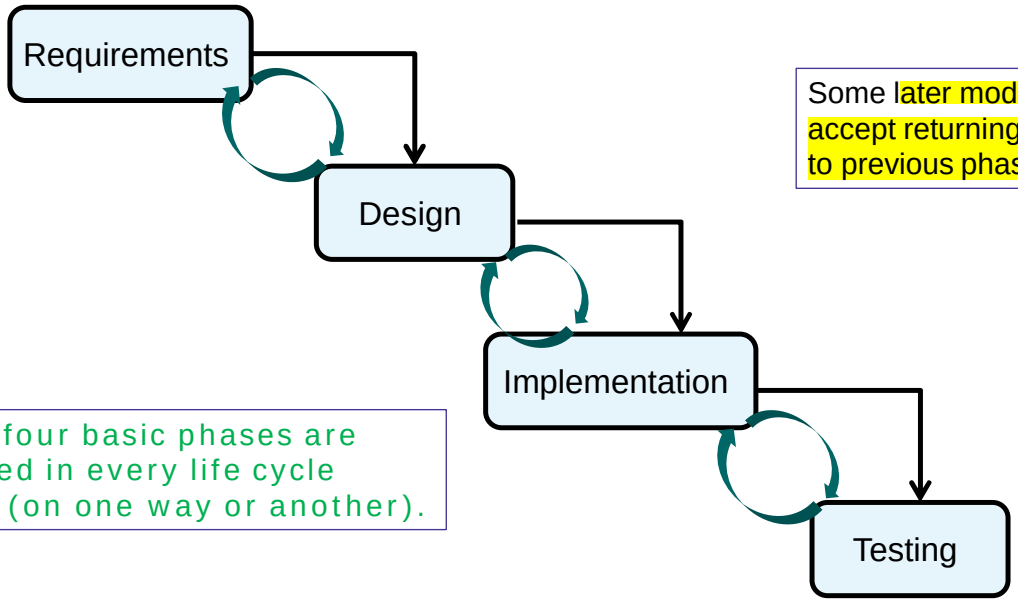


System life cycle [Sommerville SE10, 2015]



Tampereen yliopisto
Tampere University

Metaphor of the waterfall



Some later models accept returning back to previous phase.

These four basic phases are included in every life cycle model (on one way or another).

4 PHASE in basic model : Requirement => Design => Implementation => Testing

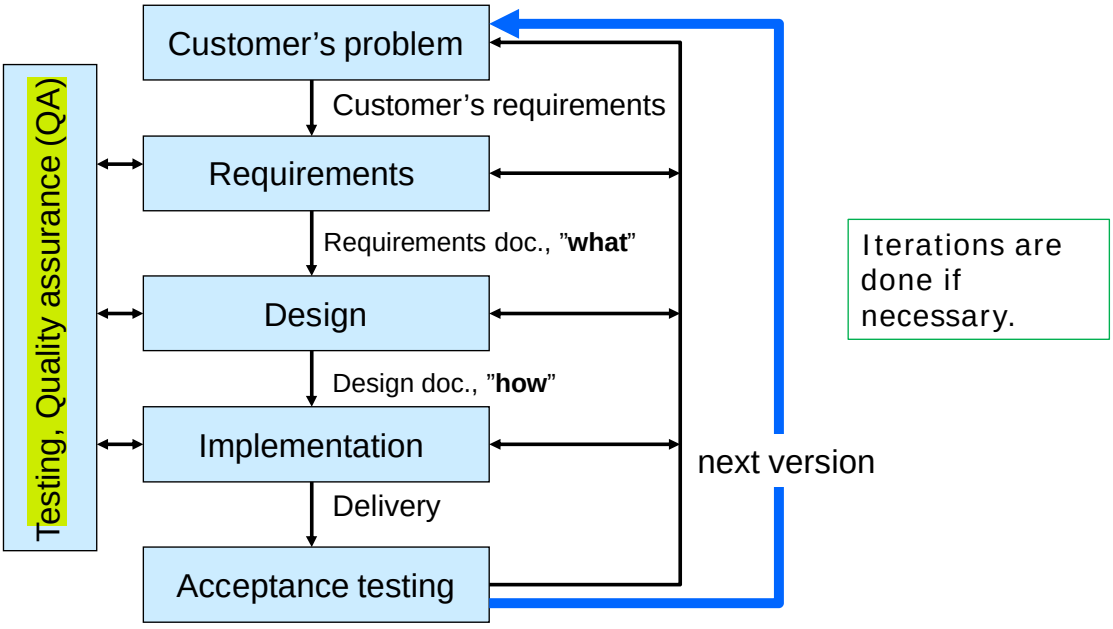
software (system) projects

Software project may be

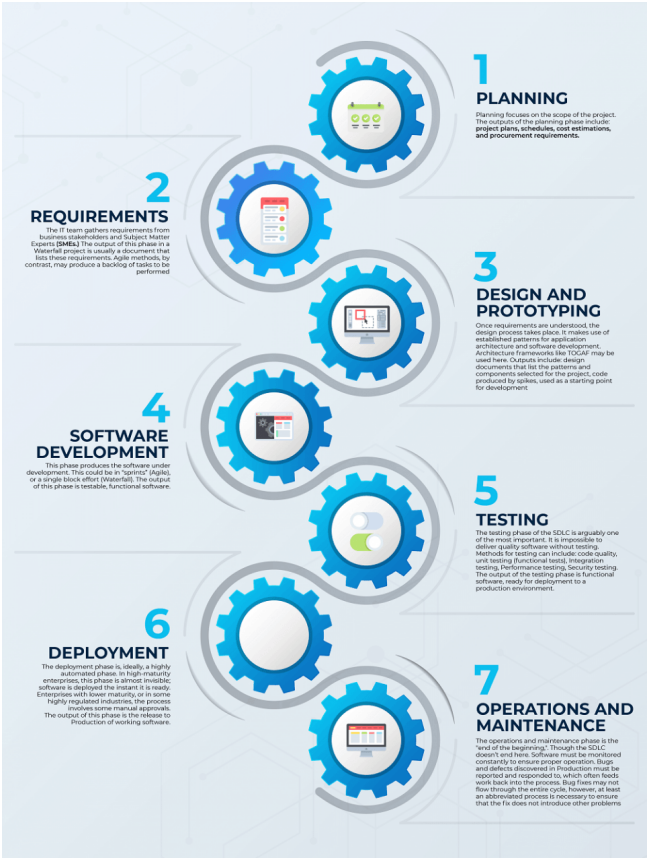
- building a new system from scratch
- enhancement of an existing system (e.g. adding features, refactoring)
- re-development, re-designing, replacing an existing system (e.g. transferring system to a new platform, and update architecture as a side effect).

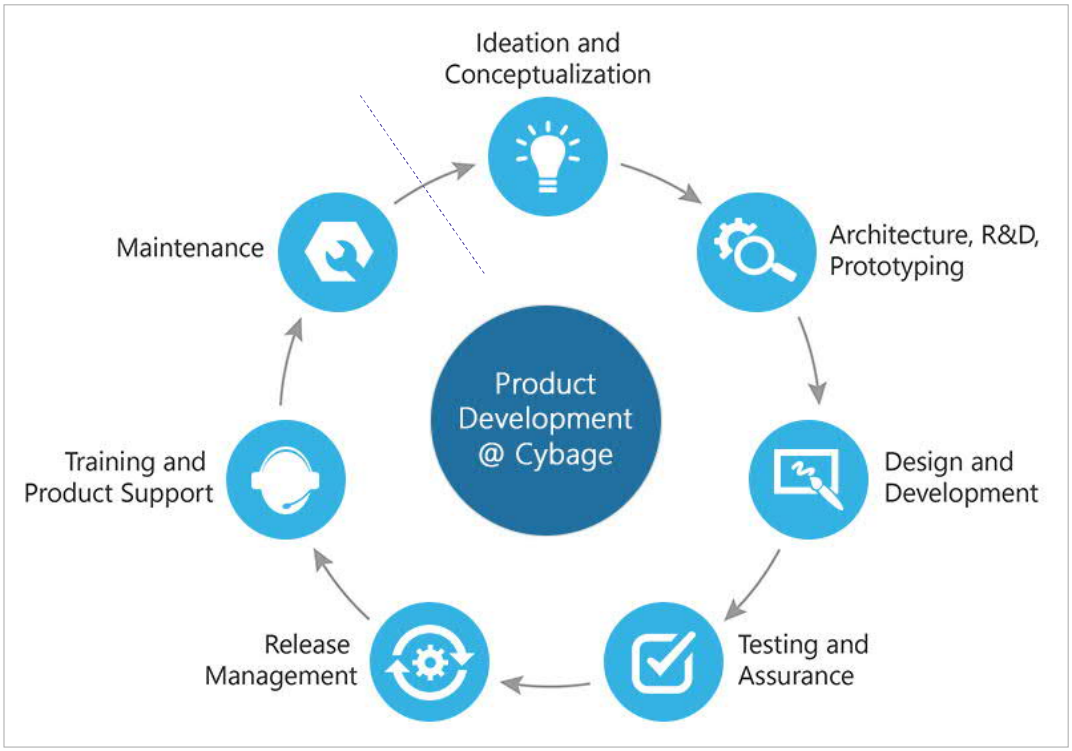
By the way, remember to involve actual end-users to project, early in the requirements phase.

Development process; there may be many variations (paths)




7 OF THE SOFTWARE
DEVELOPMENT LIFE CYCLE
PHASES
RAYGUN
BROUGHT TO YOU BY
RAYGUN
www.raygun.com


1
PLANNING
Planning focuses on the scope of the project. The outputs of the planning phase include: project plans, schedules, cost estimations, and procurement requirements.
2
REQUIREMENTS
The IT team gathers requirements from business stakeholders and Subject Matter Experts (SMEs.) The output of this phase in a Waterfall project is usually a document that lists these requirements. Agile methods, by contrast, may produce a backlog of tasks to be performed
3
DESIGN AND PROTOTYPING
Once requirements are understood, the design process takes place. It makes use of established patterns for application architecture and software development. Architecture frameworks like TOGAF may be used here. Outputs include: design documents that list the patterns and components selected for the project, code produced by spikes, used as a starting point for development
4
SOFTWARE DEVELOPMENT
This phase produces the software under development. This could be in "sprints" (Agile), or a single block effort (Waterfall). The output of this phase is testable, functional software.
5
TESTING
The testing phase of the SDLC is arguably one of the most important. It is impossible to deliver quality software without testing. Methods for testing can include: code quality, unit testing (functional tests), integration testing, Performance testing, Security testing. The output of the testing phase is functional software, ready for deployment to a production environment.
6
DEPLOYMENT
The deployment phase is, ideally, a highly automated phase. In high-maturity enterprises, this phase is almost invisible; software is deployed the instant it is ready. Enterprises with lower maturity, or in some highly regulated industries, the process involves some manual approvals. The output of this phase is the release to Production of working software.
7
OPERATIONS AND MAINTENANCE
The operations and maintenance phase is the "end of the beginning." Though the SDLC doesn't end here. Software must be monitored constantly to ensure proper operation. Bugs and defects discovered in Production must be reported and responded to, which often feeds work back into the process. Bug fixes may not flow through the entire cycle, however, at least an abbreviated process is necessary to ensure that the fix does not introduce other problems


Ideation and Conceptualization
Architecture, R&D, Prototyping
Design and Development
Testing and Assurance
Release Management
Training and Product Support
Maintenance
Product Development @ Cybage

Tampereen yliopisto
Tampere University

Software development, is much more than just coding

Software development projects vary e.g. in length and purpose.

Koronavilkku is mobile application which was developed in three months (open source in GitHub). APIs and some components were already existing.

Apotti project is five years long, and is currently in use, still some development to be done.

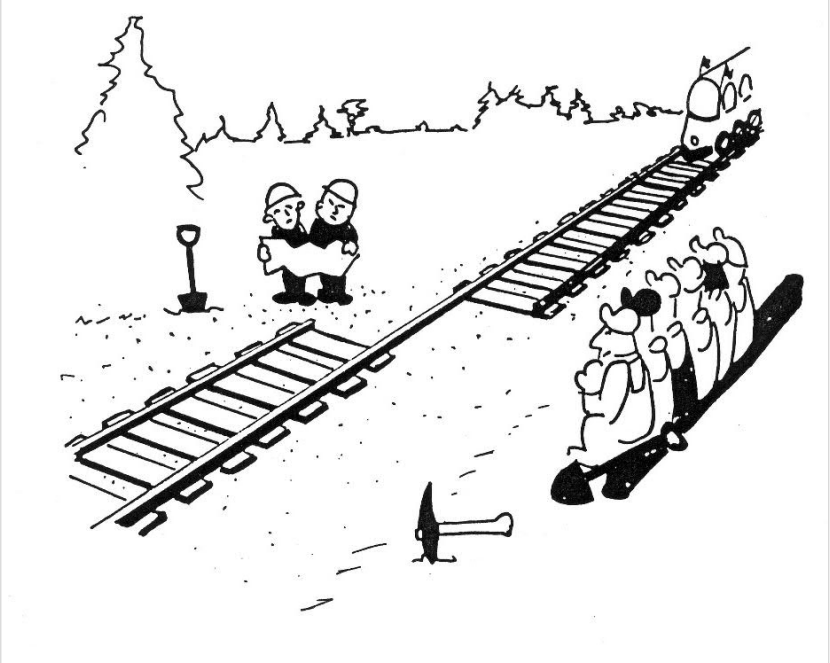
Olkiluoto power plant (OL 3) is still having some automation system testing to be done. It was to started 2005, was to be ready 2009, but will probably start 02/2022, due mechanical problems.

Tampereen yliopisto
Tampere University

nice planning... but not enough

For understanding requirements right, it would be good to show current demos (GUI or functioning skeleton program) to customer every now and then (every 2..4 weeks), to get feedback.

A Finnish study from large-scale international project revealed that unofficial communication between vendor and customer (in seminars etc.) was of great help.



communication btw vendor & customers

Tampereen yliopisto
Tampere University

Teamwork

Tampereen yliopisto
Tampere University

Teamwork is essential

Because of modern software size and complexity, one can not produce commercial software by his/her own.

One developer can make a suitable program for personal use, but any commercial software sooner or later needs additional workforce.

One developer can master a few tools and technologies, but nobody can e.g. all popular programming languages.

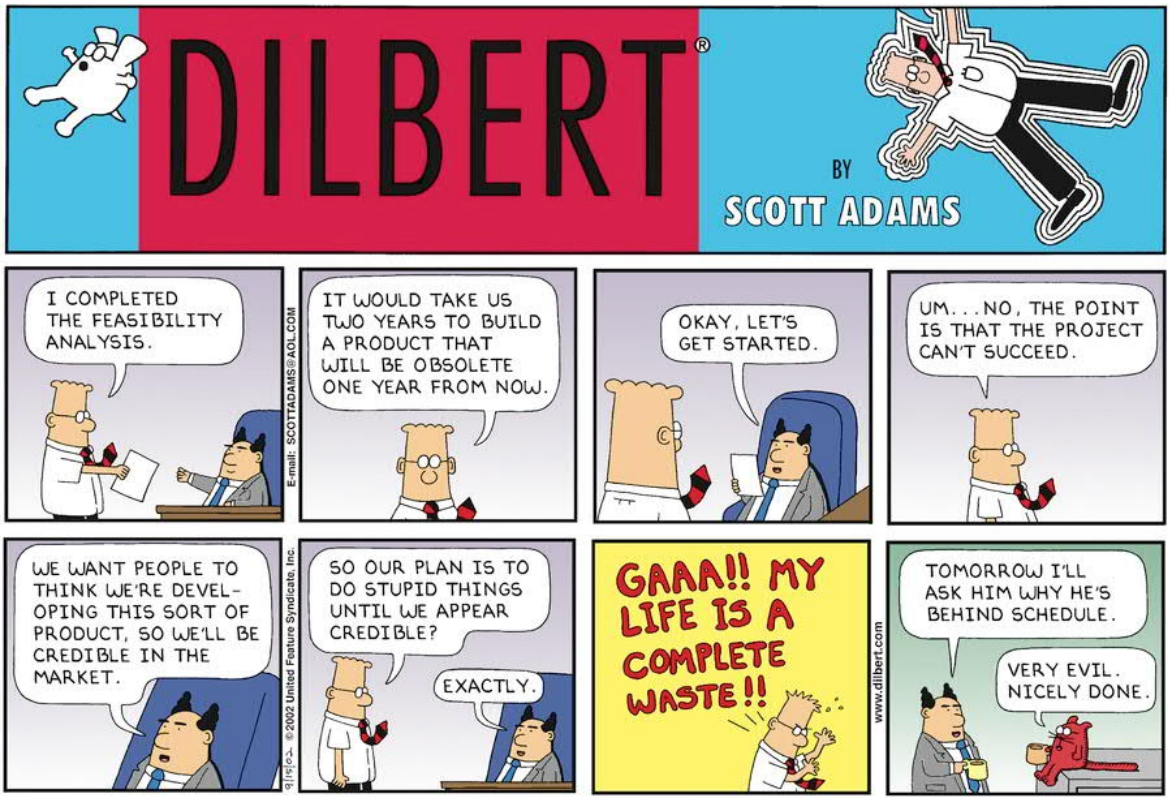
BTW. It is better to learn any one programming language well, after that learning other languages are easier (said by company recruiters). There is not much use for developer who can several languages poorly, but knows no one well. The same suit to methods and methodologies (set of methods).

Feasibility study / preliminary analysis

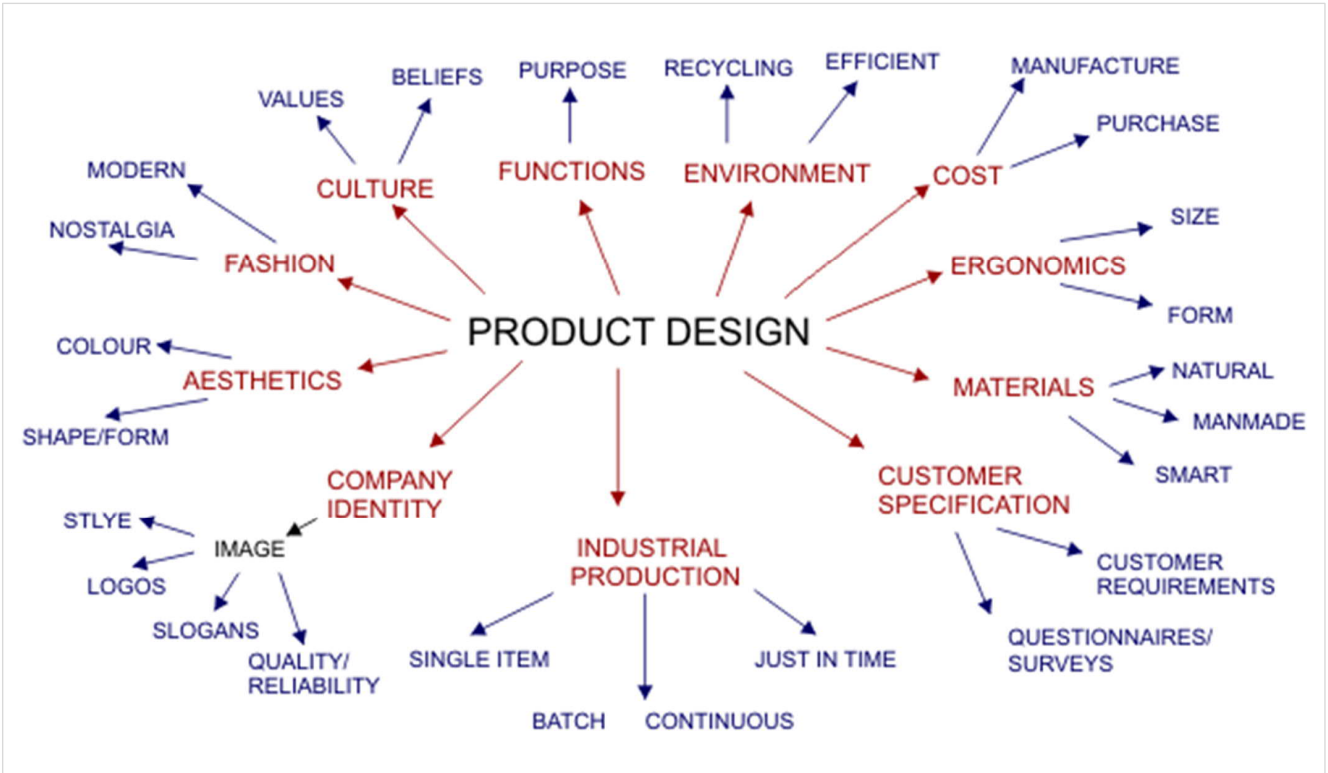
Feasibility Study, Preliminary Analysis ("reality check")

- usually made by the vendor/developer (sw company)
- is made at the very beginning, after the idea of a new system
- a few hours … a few working days is spent
- is made to find out whether a project will be started, or not
- do we have enough time to do such a project, skilled workers, tools, money, possible other/further customers in sight,…
- in many cases the result is that such a project is not started (perhaps later e.g. when technology will be more mature)
- if project is started, matters of Feasibility Study will be described in more detail in Requirements Specification and Project Plan.

BTW: the first thing is to check internet if such thing has already been done ! "Do not re-invent the wheel."



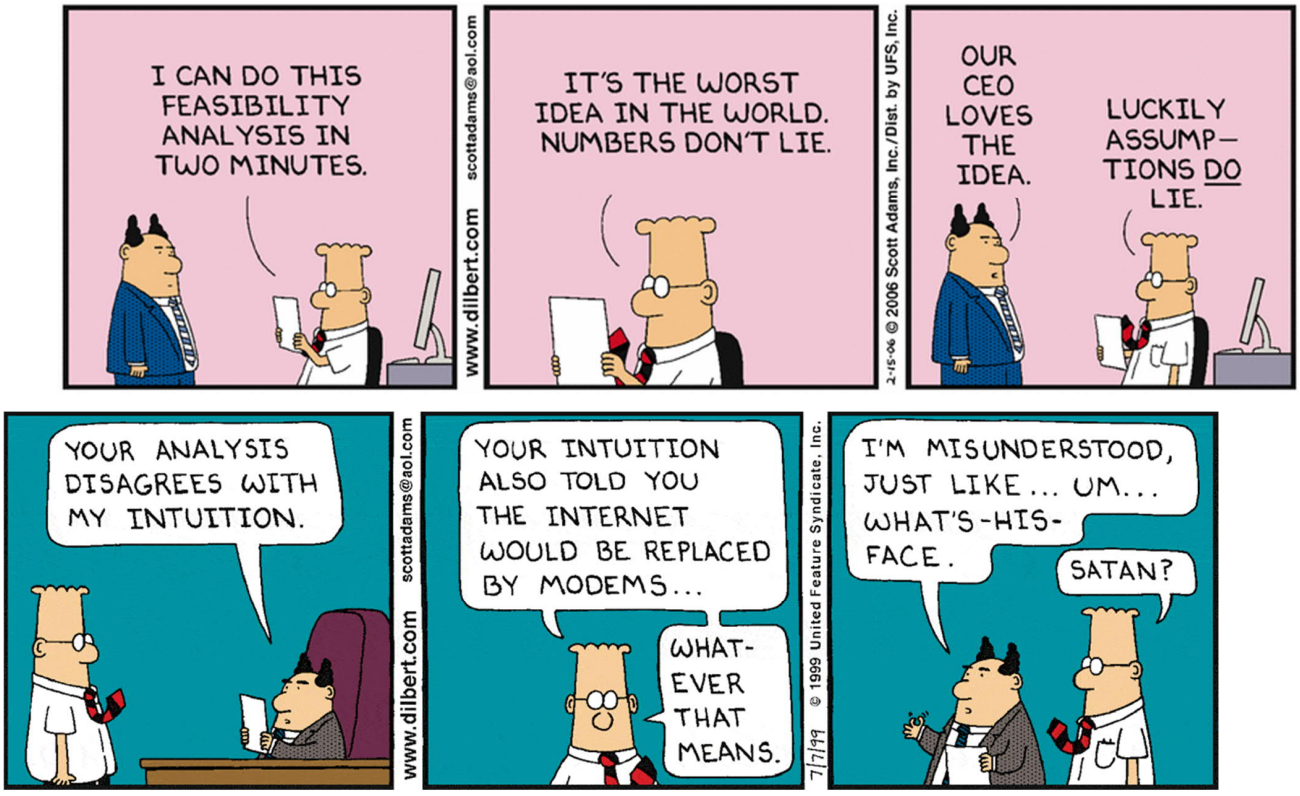
Forget nothing



[http://dnt-thoughts.blogspot.com/2010/06/development-update.html]

Tampereen yliopisto
Tampere University

Real data vs. feelings ?



Tampereen yliopisto
Tampere University

Start for requirements, one path

- customer has some ideas and thoughts about requirements
- developer company writes first basic requirements (based on customer's data)
- developer makes stakeholder search and analysis (which to include)
- developer (re-)defines basic requirements, GUI sketch
- requirements are discussed between customer and developer
- developer writes more concrete requirements (more details), GUI proto
- end-users are taken into account, GUI proto 2
- requirements and GUI are defined in detail
- requirements are discussed in detail between customer and developer
- requirements specification document (includes GUI)

...requirements alternate / interchange / vary at least slightly during the project...

Tampereen yliopisto
Tampere University

Stakeholder analysis

[ISO 21500:2012]

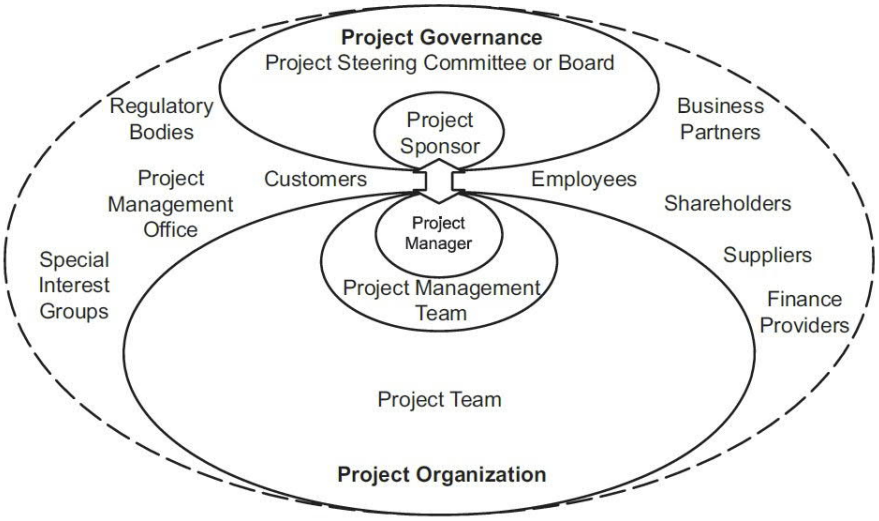
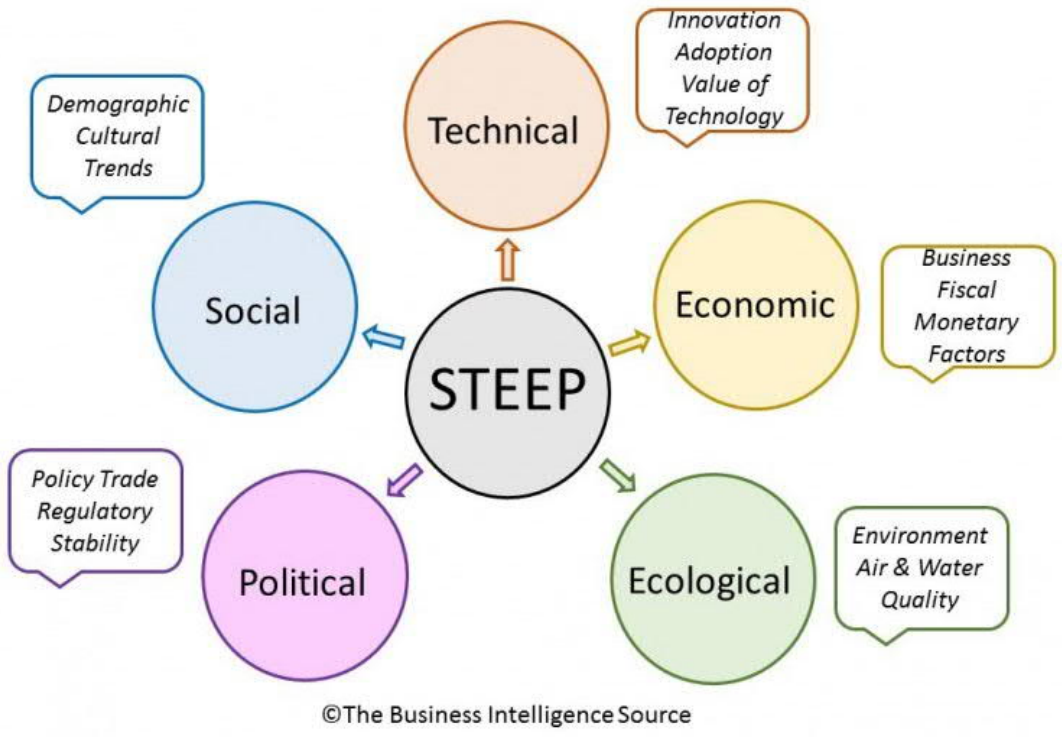


Figure 4 — Project stakeholders

STEEP = Social Technical Economic Ecological Political



PESTLE… STEPLE… stakeholders

The following illustrates the PESTLE analysis of Microsoft dated back in 2011:

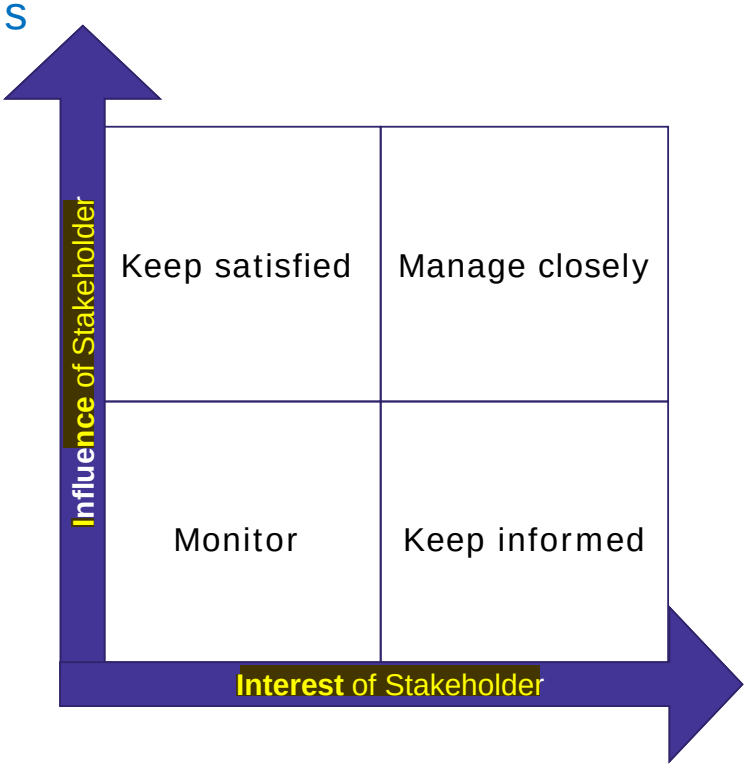
Table 1-2: PESTLE analysis of Microsoft in 2011

Political • Currency and taxation policies in different countries could significantly increase cost of business operations. • Weak intellectual property protection in emerging markets impairs revenue from such markets.	**Economic** • After-effects of the 2008 economic crisis affect customer demand. • Exchange rate fluctuation makes global pricing strategies more difficult.	**Social** • Increasing awareness of open technologies and business ethics impair the reputation of the Microsoft brand. • Lack of mobile presence causes people to relate Microsoft as an outdated and less innovative brand.
Technological • Mobile, cloud-based and service-based services are gaining more popularity but Microsoft's offerings are not good enough. • Time-to-market of competitors has been shortened significantly. • Competitors' solutions do better on end-user usability and "coolness" (e.g. Apple iPhone)	**Legislative** • Legal disputes on monopoly and abusive practices. • Legal disputes on patent infringements. • Privacy concerns on cloud-based products.	**Environmental** • Power consumption of data centres • Eco-concerns on product packaging materials.

Stakeholders

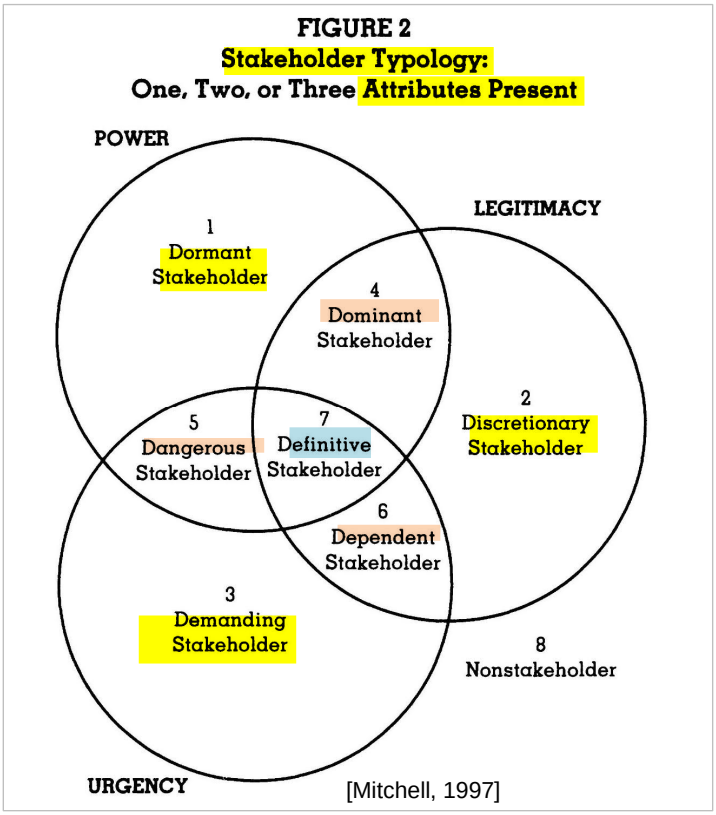
At this matrix top right corner surely contains the most important stakeholders.

But depending on the project, also other groups may and should be taken into account, when finding out requirements.



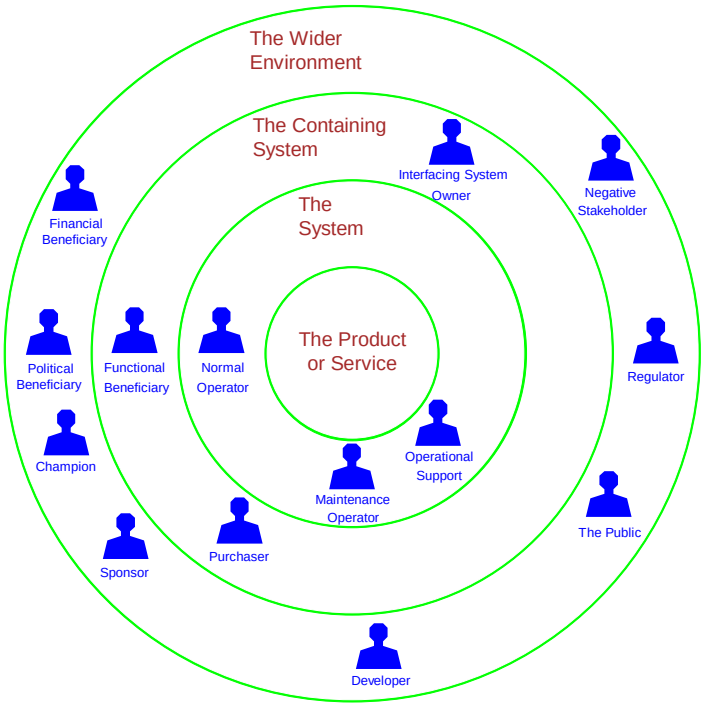
Stakeholders

"Important stakeholders are all who have possibilities and will to affect the project."



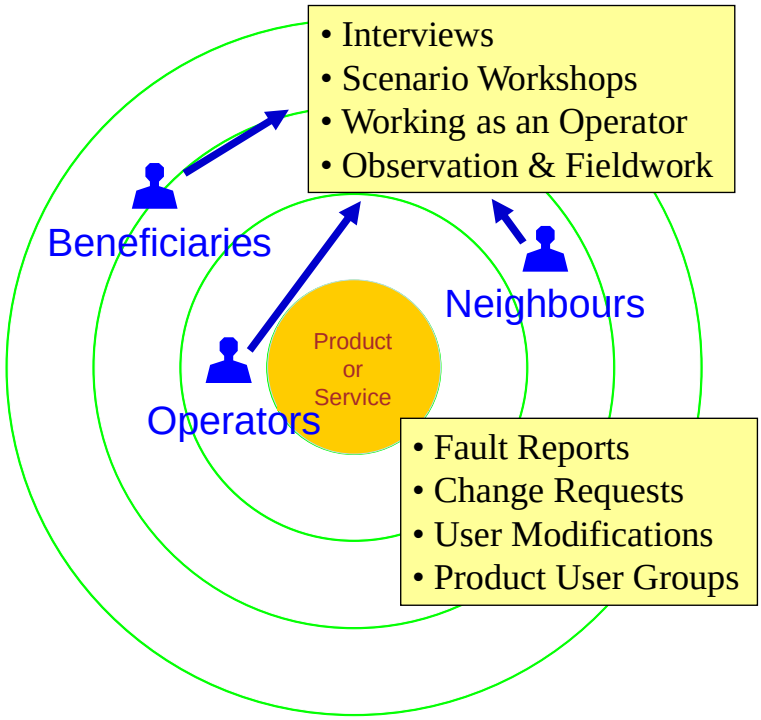
[Mitchell, 1997]

Stakeholders in a Typical System



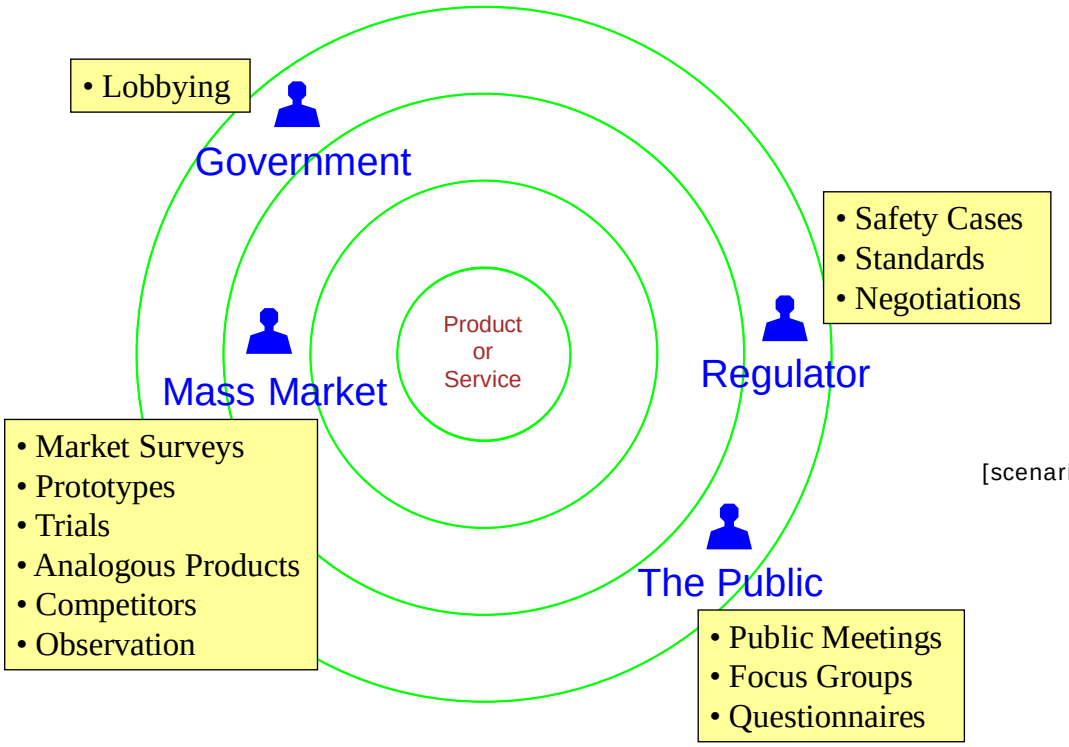
[scenarioplus.org.uk]

Discovery Contexts and Sources



[scenarioplus.org.uk]

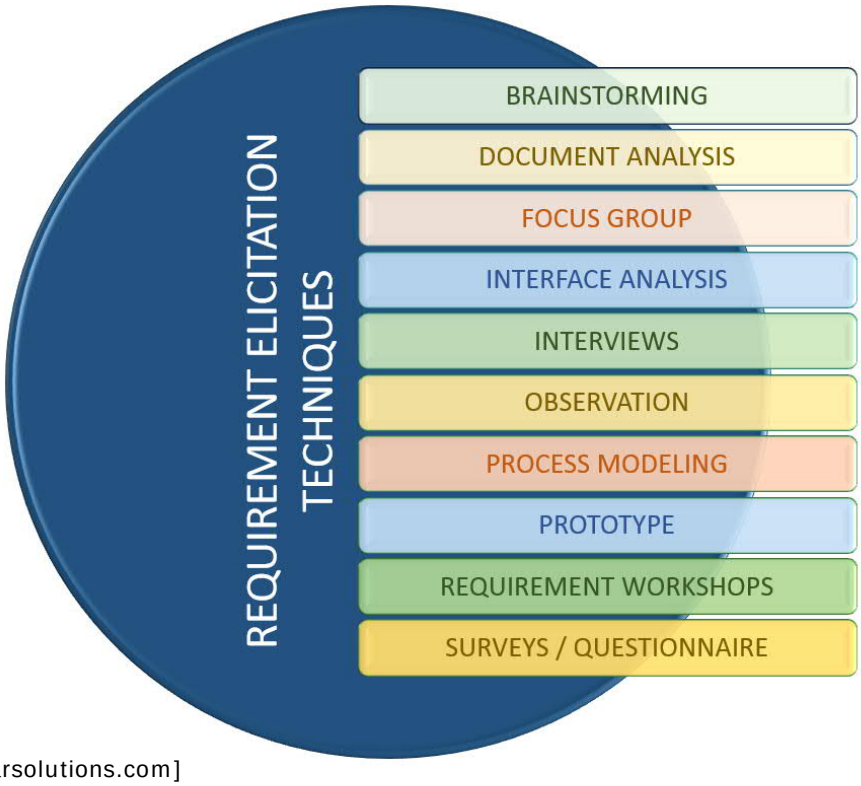
Discovering from Non-Operational Roles



[scenarioplus.org.uk]

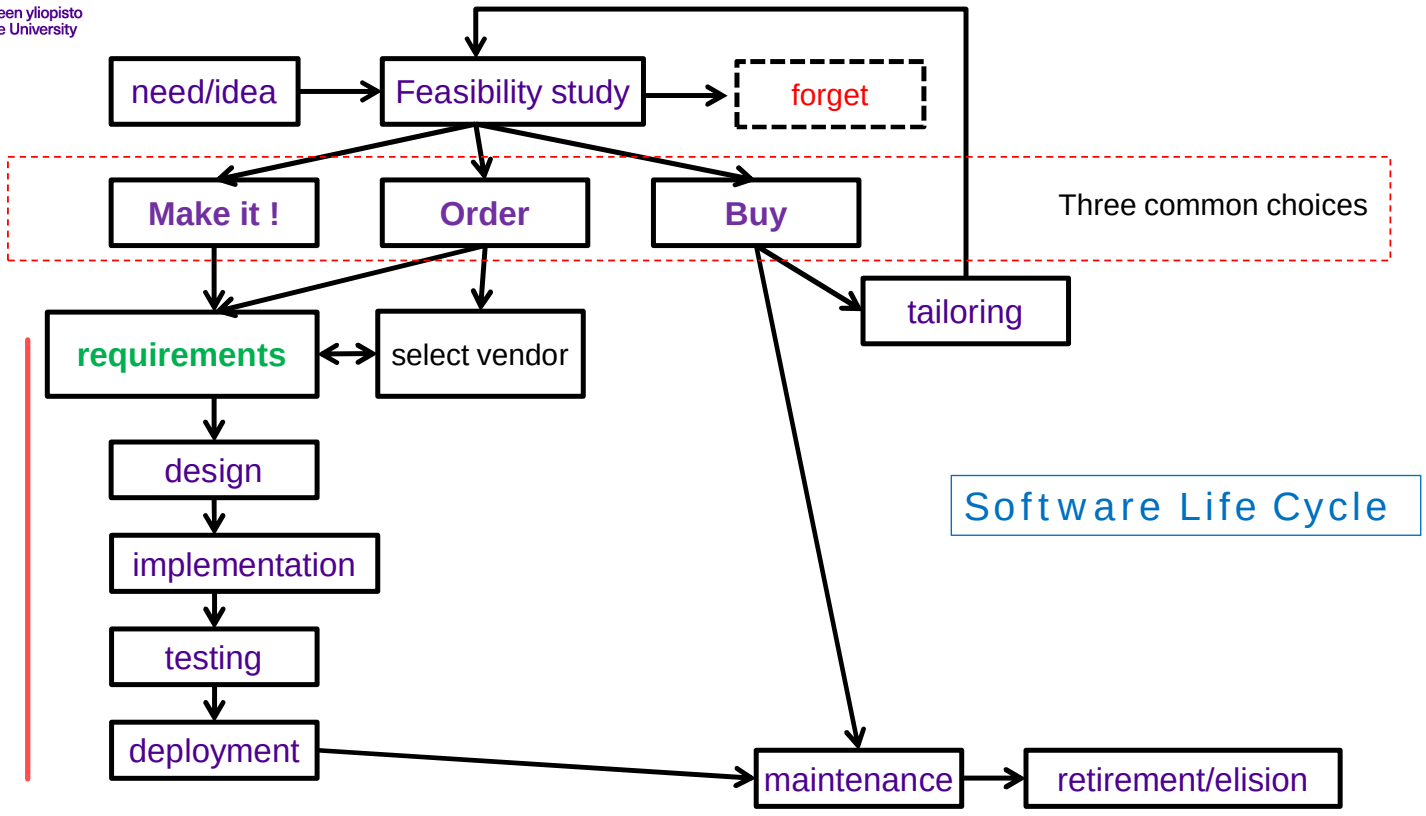
Tampereen yliopisto
Tampere University

Some requirement elicitation techniques



[www.anarsolutions.com]

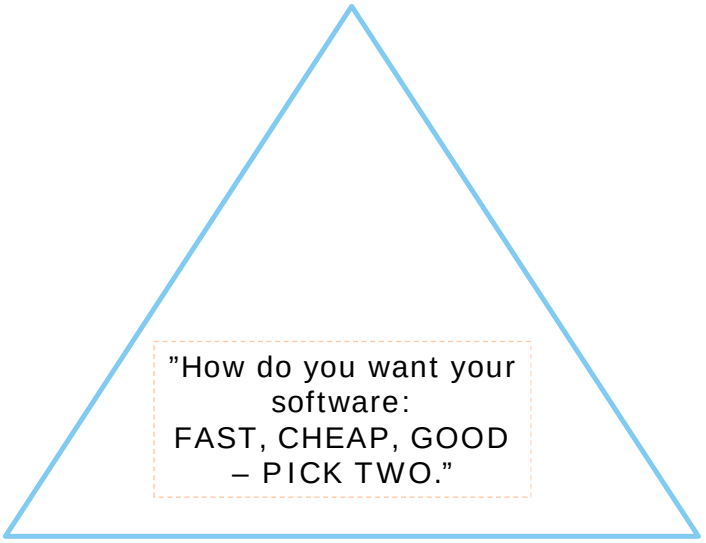
How to get software product



Software Life Cycle

Tampereen yliopisto
Tampere University

”Iron triangle” of SW projects



Triangle is always some kind of a compromised balance between the three factors.

Remember also the fourth quality factor in business: SERVICE.

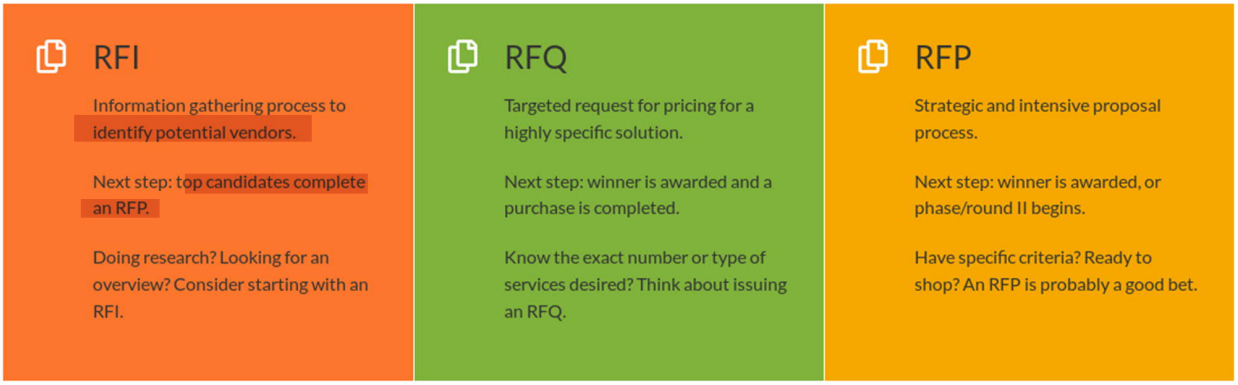
Tampereen yliopisto
Tampere University

Sw development ”process”

Customer gets an idea for a new information system (or replacing an old one by a new and better).

Customer writes down most wanted/needed requirements (functionalities/features), hopefully in a list of priority order (usually in spreadsheet).

If not COTS (commercial off the shelf), customer makes RFI/RFQ/RFP (request for information / quotation / proposal). There are many templates for such documents.

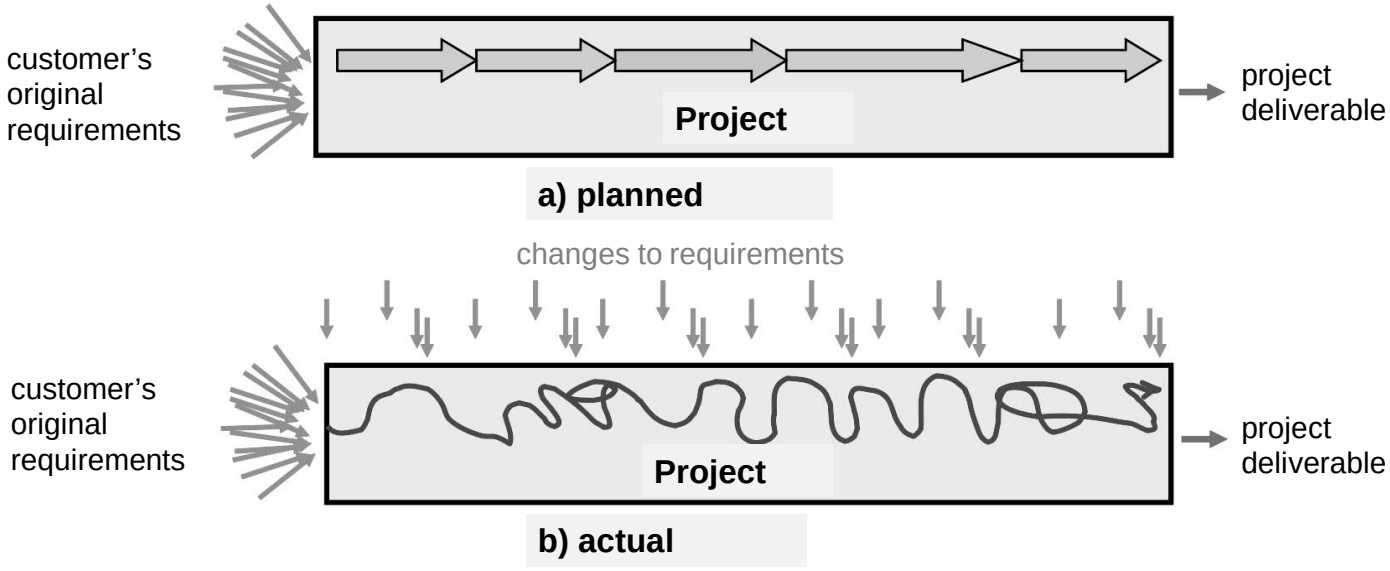


[rfp360.com]

GEt idea for new information system (or replacing old) => write down wanted/needed => RFI/RFQ/RFP

Tampereen yliopisto
Tampere University

SW project is reacting to changes and (hopefully small) surprises



(compare that to your planned study path…?)

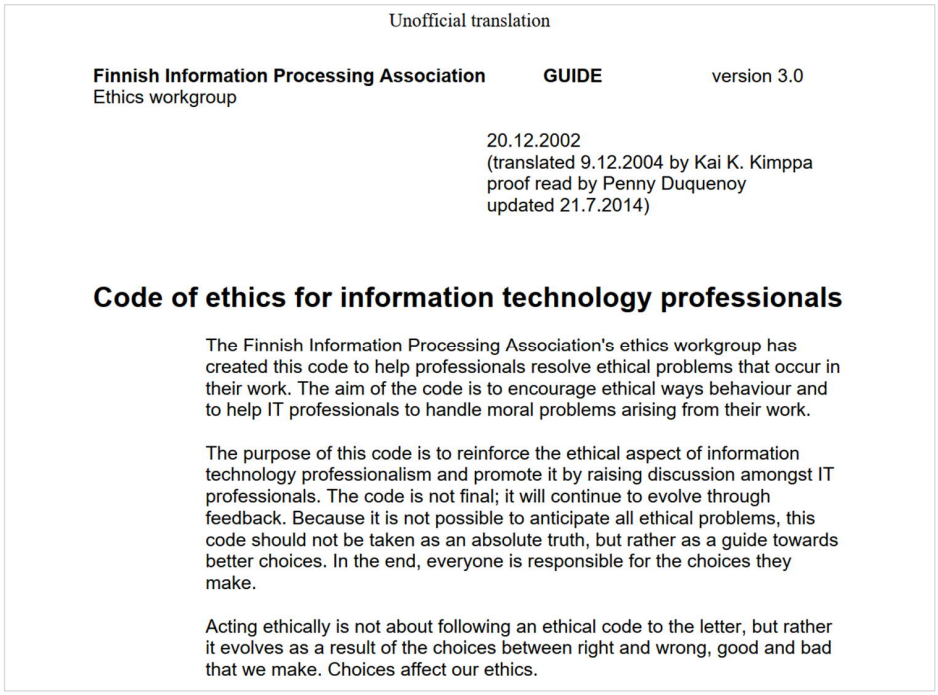
Tampereen yliopisto
Tampere University

Ethics

Code of ethics in software engineering

https://tivia.fi/in-english/

- Authority and responsibility
- Knowledge and experience
- Attitude
- Communications
- Consequences of professionals' work
- Other people
- Promotion of ethics.



Unofficial translation

Finnish Information Processing Association **GUIDE** version 3.0
Ethics workgroup

20.12.2002
(translated 9.12.2004 by Kai K. Kimppa
proof read by Penny Duquenoy
updated 21.7.2014)

Code of ethics for information technology professionals

The Finnish Information Processing Association's ethics workgroup has created this code to help professionals resolve ethical problems that occur in their work. The aim of the code is to encourage ethical ways behaviour and to help IT professionals to handle moral problems arising from their work.

The purpose of this code is to reinforce the ethical aspect of information technology professionalism and promote it by raising discussion amongst IT professionals. The code is not final; it will continue to evolve through feedback. Because it is not possible to anticipate all ethical problems, this code should not be taken as an absolute truth, but rather as a guide towards better choices. In the end, everyone is responsible for the choices they make.

Acting ethically is not about following an ethical code to the letter, but rather it evolves as a result of the choices between right and wrong, good and bad that we make. Choices affect our ethics.

Ethics, Code of Conduct

IEEE = Institute of Electronics and Electrical Engineers [I-triple-E]

IEEE CODE OF CONDUCT

Approved by the IEEE Board of Directors, June 2014

1. Be respectful of others
2. Treat people fairly
3. Avoid injuring others, their property, reputation or employment
4. Refrain from retaliation (= no revenge)
5. Comply with applicable laws in all countries where IEEE does business and with the IEEE policies and procedures.

The IEEE Code of Conduct describes IEEE members' and staff's commitment to the highest standards of integrity, responsible behavior, and ethical and professional conduct.

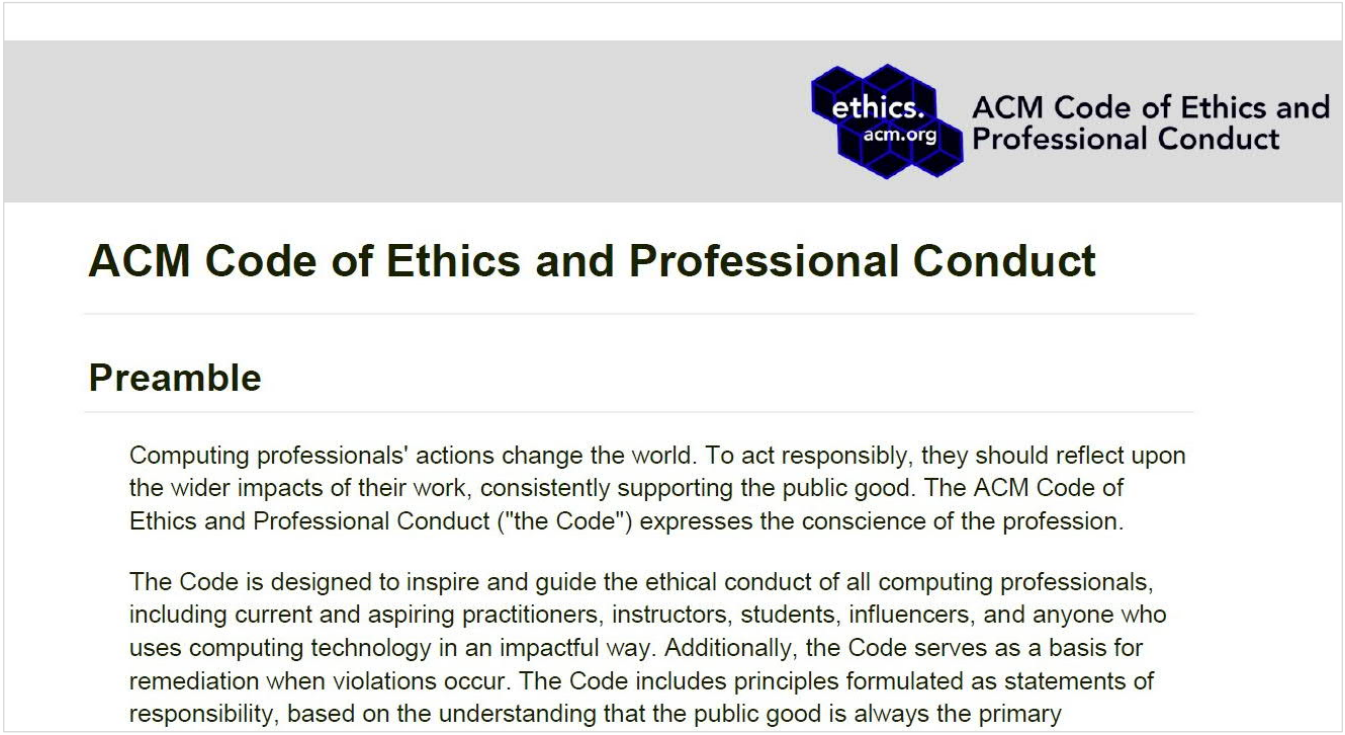
Tampereen yliopisto
Tampere University

Ethics, Code of Conduct

Simplified ethics

- take care of yourself – heroic effort as a ”suicide coder” is no good in long run
- respect yourself
- respect (be polite to) your fellow workers
- respect developer/vendor company
- respect customers and customer company
- many problems and conflicts can be solved by discussion
- if you see/discover some error in project, inform others about it (do not hide)
- develop yourself, continuous life-long learning (you can not avoid that).

Tampereen yliopisto
Tampere University



Tampereen yliopisto
Tampere University

https://www.entrepreneur.com/article/311410

ETHICS

Why Tech Companies Need a Code of Ethics for Software Development

With so much potential for software to go bad, it's important that developers commit to doing good.

Add to Queue

NEXT ARTICLE

Dave West
CEO and Product Owner, Scrum.org

April 19, 2018 6 min read

ETHICS
What Working at Enron Taught Me About Corporate Ethics

ETHICS
Are You in it for 5 Years or 50? The Trust You Earn Will Determine How Long Your Business Lasts.

ETHICS
Only Ethical Marketing Will Stand the Test of Time

ETHICS
Can a Vegan Fashion Brand Survive in the Market?

NEPOTISM
5 Simple Ways to Handle Nepotism in the Workplace

Tampereen yliopisto
Tampere University

Documentation

Tampereen yliopisto
Tampere University

Documentation (don't be scared)

- Every serious software has some documentation.
- But the amount of documentation should be kept reasonable, that means minimum… short.
- Good documentation is complete and unambiguous but short. Have you seen any ?
- At least users need some guide/manual (at least on-line help (F1), or ”readme.txt” or ”help text” file).
- Some kind of maintenance guide/manual would help developers make later fixes/corrections, adding or deleting parts or functionalities. Important document !
- In longer projects weekly or bi-weekly reports give good information to stakeholders.

Tampereen yliopisto
Tampere University

The long path of documentation may be:

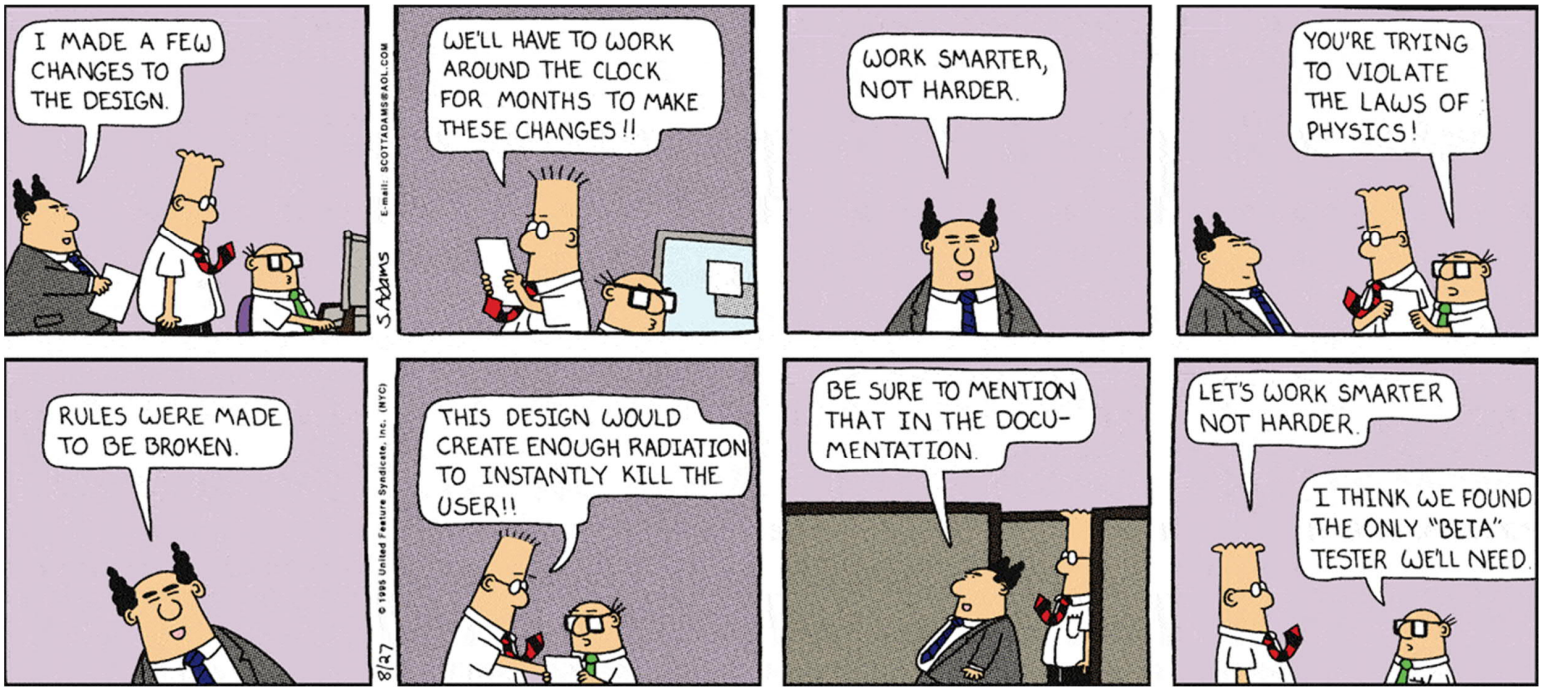
- Feasibility Study / Preliminary Analysis (is the project worth starting ?)
- Project Plan (updating depends of use and purpose)
- Requirements (SRS = Software Requirements Spefication)
- Design (architecture, detailed/module)
- GUI sketches/designs/wireframe (Graphical User Interface)
- Test Plan (module, integration, system, acceptance)
- code comments (follow Coding Conventions / Style Guide)
- User Manual / User Guide, or Getting Started card / Quick Reference Guide
- Maintenance Guide (for developer company, IMPORTANT DOCUMENT)
- Test Report (system, acceptance)
- Project Final Report / Lessons learnt.

Not all these are made in the same project; need for documentation depends of the project and customer.

Some documentation is always needed in commercial sw projects. E.g. for maintenance work.

Technical debt; e.g. you have made some ”shortcuts”, perhaps some nice ”hacks” to deliver product quickly, but later at maintenace phase that causes troubles.

Tampereen yliopisto
Tampere University



Reuse

Reuse, ”do not invent wheel again”

For many (tens of) years it has been worried that reuse in software development projects should be more common practice.

In product families reuse may well be more than 50 % of code.

You may reuse e.g. code and user stories.

Code reuse may be

- full reuse, reuse component as it is
- modified reuse, some modifications to component has to be made.

Why reuse ?

- less work (= costs)
- quick development (to market)
- better quality (already tested components).

BUT: it is said that writing reusable code is 50 % slower than ”normal” work. That means 1,5 times working hours.

reuse, there could be more of that

Why ”invent a wheel”, if it has already been done earlier ?

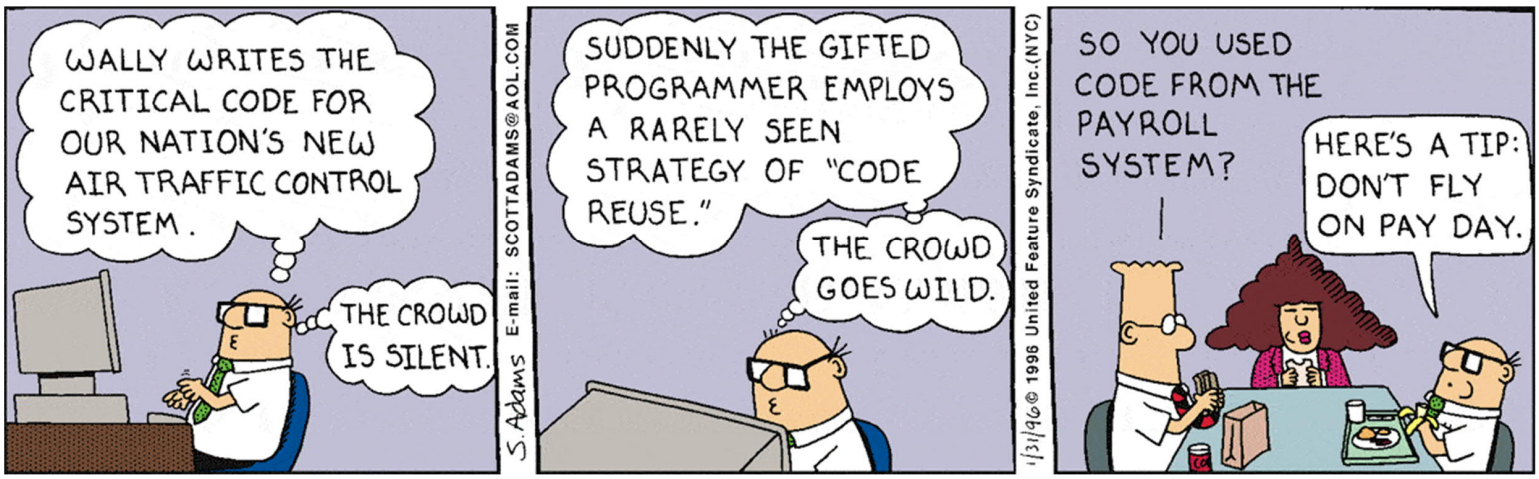
Reuse is something that is not yet fully utilised at software industry.

Ideally, some 50 % of code could be reused, but the actual amount is something like 15..30 %. Well, of course if you are making similar product to several customers, you may reuse more than 50 % of the code (which is reasonable).

Two major blocks for reuse are thinking or acting in a wrong way;

- ”Nobody has done this before, it takes me just a few days to code...”
- ”Somebody may have done this earlier, but where the code is and how can I find it... and if I find it, how to find out how it works...”

On the other hand, if you reuse working code, you still need to test it in the new product !



Work also as what to consider when selecting reusable component

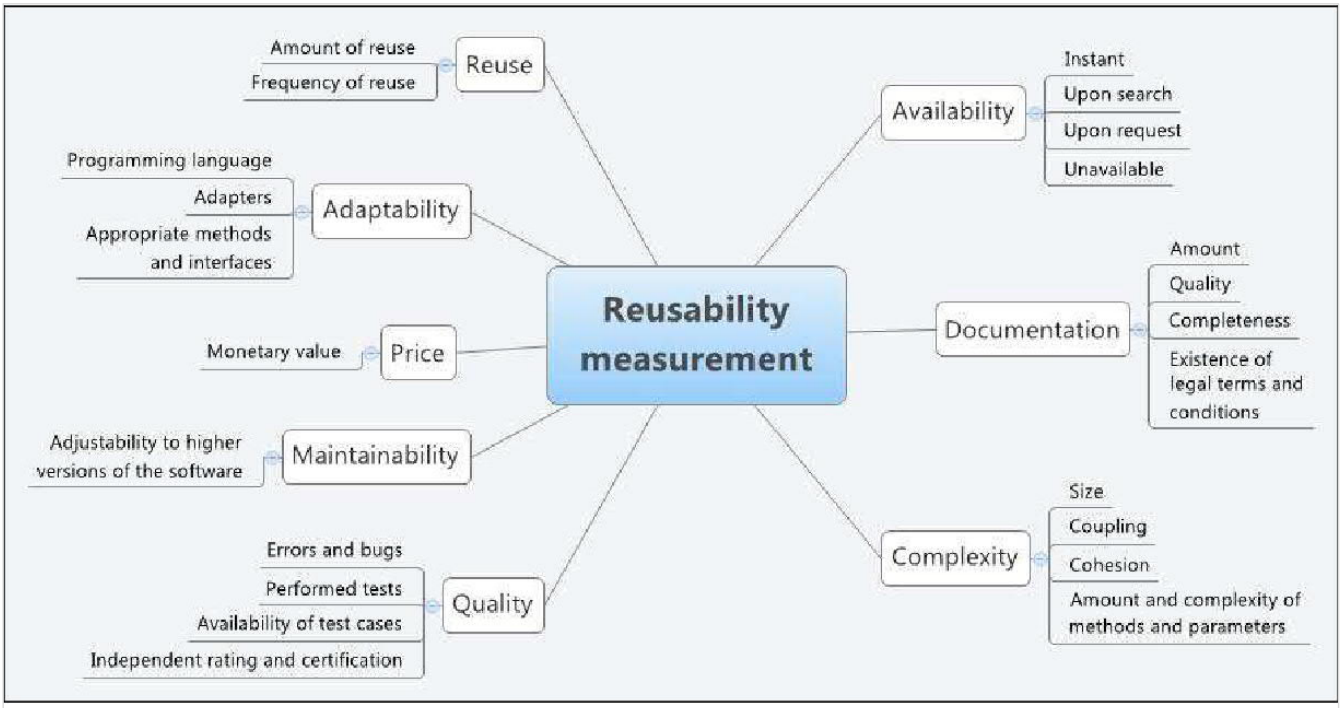


Figure 1. Factors influencing reusability measurement.

[Structuring Software Reusability Metrics for Component-Based Software Development, 2012]

Highlights - What to remember

- at every process model there is some amount of: requirements – design-implementation – testing
- today's aeroplanes and even cars have millions lines (MLOC) of code (how an earth somebody can handle such a mess… magic… good luck… ? See forthcoming lectures)
- software projects are not scalable; what is OK for small project, may not succeed in large project
- as a developer, remember to involve end-users
- as a customer, remember to involve actual end-users
- some amount of documentation needs to be done, at least for maintenance
- always check first if something similar has been already made (e.g. one hour search)

- there may be problems as in SISU; there are hundreds of different variations how end-users use (and think their work) study systems. Solution: modify work process, not software ?!?

Tampereen yliopisto
Tampere University

Now the additional L2 extra slides are here

No time to show these at lectures, but otherwise good to know, at least if you are a major reader.

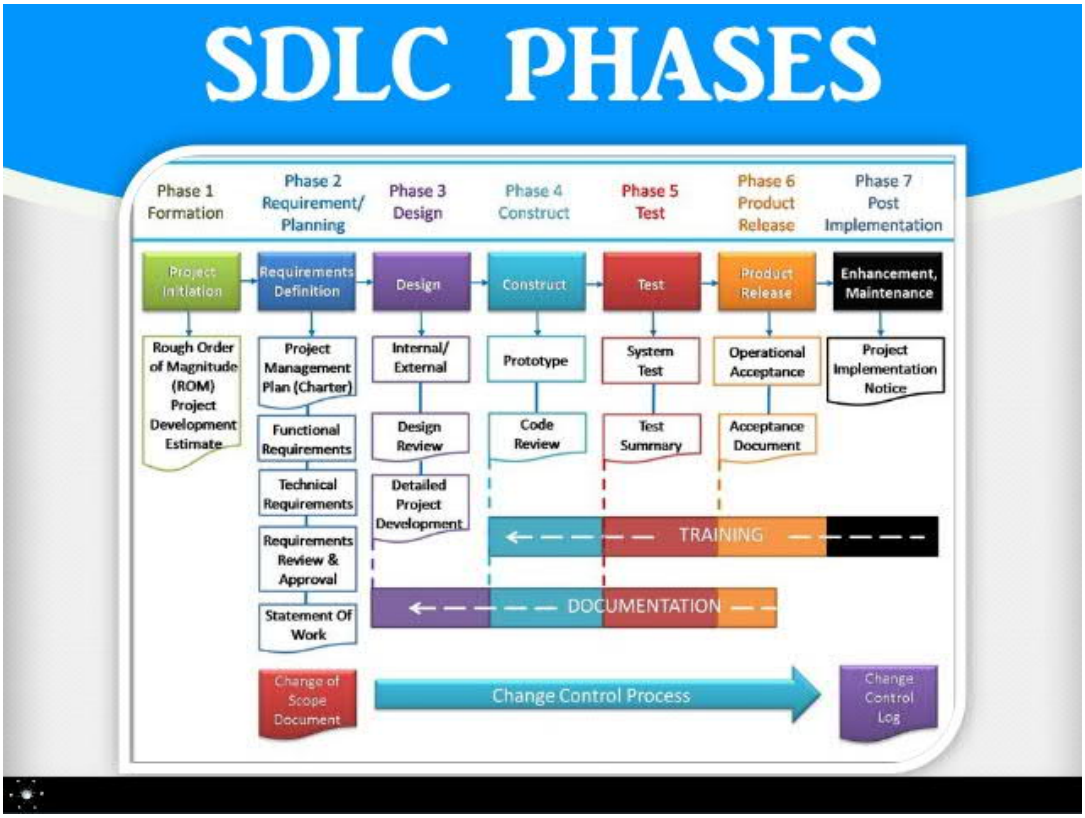
Tampereen yliopisto
Tampere University

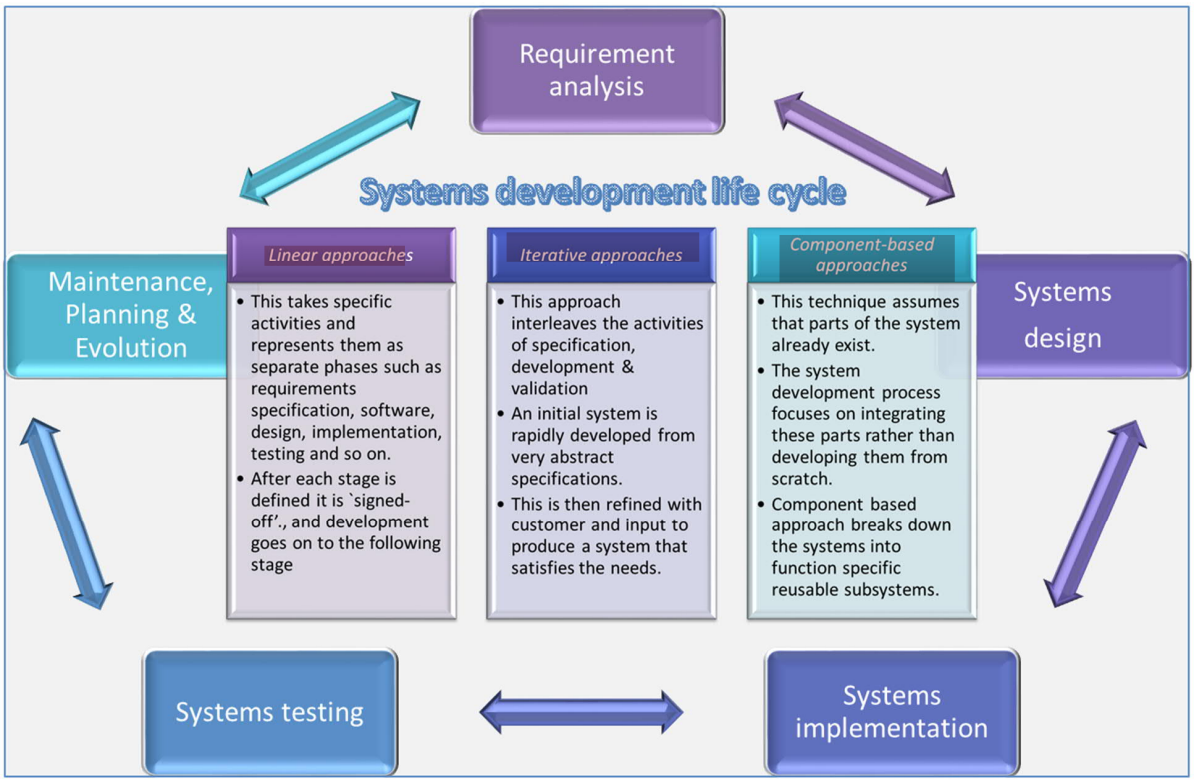
Now the additional L2 extra slides are here

No time to show these at lectures, but otherwise good to know, at least if you are a major reader.

Now the additional L2 extra slides are here

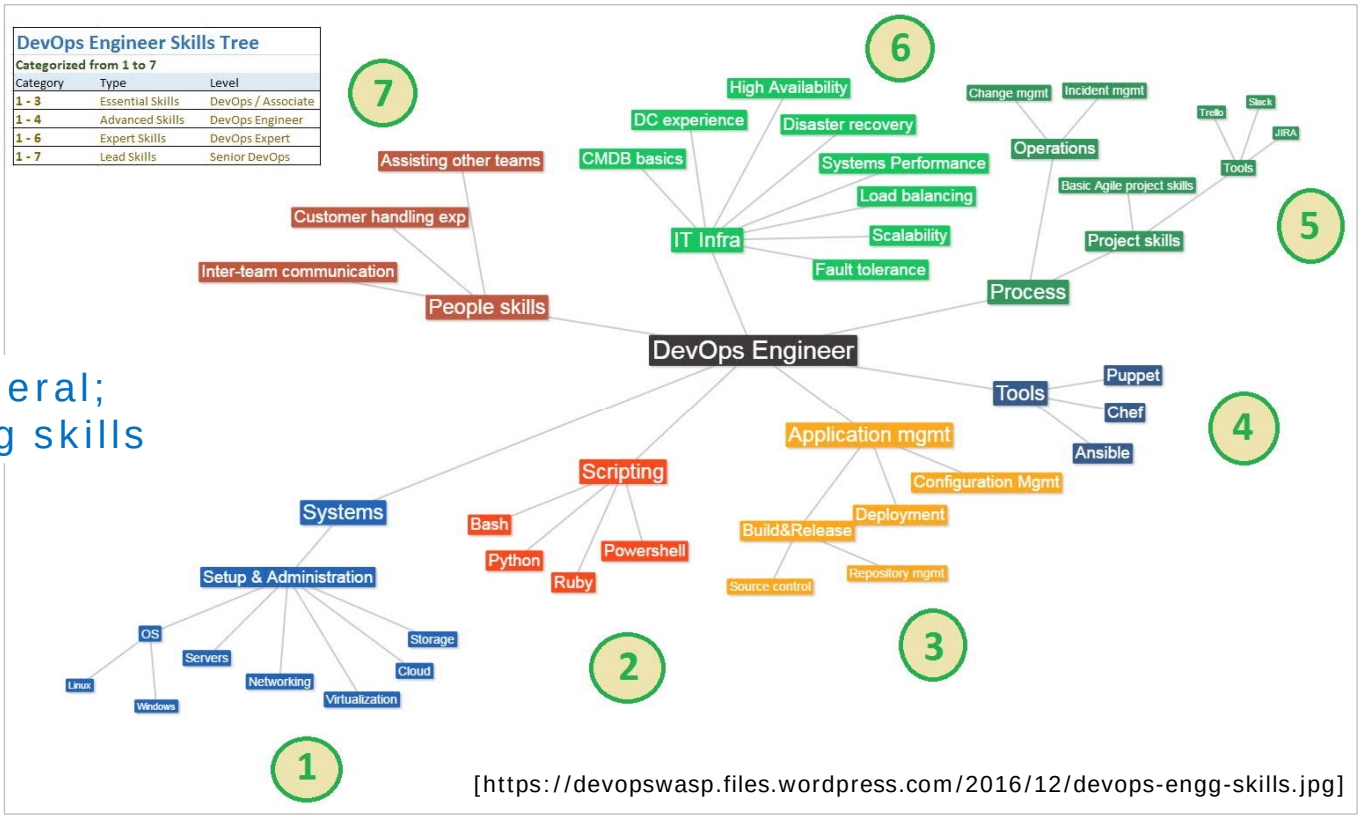
No time to show these at lectures, but otherwise good to know, at least if you are a major reader.


SDLC PHASES
Phase 1 Formation
Phase 2 Requirement/ Planning
Phase 3 Design
Phase 4 Construct
Phase 5 Test
Phase 6 Product Release
Phase 7 Post Implementation
Project Initiation
Requirements Definition
Design
Construct
Test
Product Release
Enhancement, Maintenance
Rough Order of Magnitude (ROM) Project Development Estimate
Project Management Plan (Charter)
Functional Requirements
Technical Requirements
Requirements Review & Approval
Statement Of Work
Internal/ External
Design Review
Detailed Project Development
Prototype
Code Review
System Test
Test Summary
Operational Acceptance
Acceptance Document
Project Implementation Notice
TRAINING
DOCUMENTATION
Change of Scope Document
Change Control Process
Change Control Log


Requirement analysis
Systems development life cycle
Maintenance, Planning & Evolution
Systems design
Systems testing
Systems implementation
Linear approaches
• This takes specific activities and represents them as separate phases such as requirements specification, software, design, implementation, testing and so on.
• After each stage is defined it is `signed-off'., and development goes on to the following stage
Iterative approaches
• This approach interleaves the activities of specification, development & validation
• An initial system is rapidly developed from very abstract specifications.
• This is then refined with customer and input to produce a system that satisfies the needs.
Component-based approaches
• This technique assumes that parts of the system already exist.
• The system development process focuses on integrating these parts rather than developing them from scratch.
• Component based approach breaks down the systems into function specific reusable subsystems.

Tampereen yliopisto
Tampere University

In general; sw eng skills



[https://devopswasp.files.wordpress.com/2016/12/devops-engg-skills.jpg]

Tampereen yliopisto
Tampere University

Today's ("hype" ?) software development method(ologie)s

- agile methods (agile, FI: ketterä); iterative
- Scrum (one agile method)
- Kanban; visible board, WIP (pull), optimisation
- Lean = eliminate waste, optimise
- DevOps = DevelopmentOperations (CI, CD)
- SEMAT = Software Engineering Method and Theory
- SAFe = Scaled Agile Framework, for BIG projects
- Blended Agile = combination of two or more Agile-methods
- Hybrid Agile = Agile + non-Agile-method.

The Best Method is the one, which works best for your project.

Methodology is a set on methods (and practices). Keep your eyes and ears open for methods worth trying in your own work.

Tampereen yliopisto
Tampere University

Software system may be described by its size (not very accurate), 1 / 2

Tiny - 1000 lines or less. Easy to keep every tiny detail and every line in your head and know exactly where it is.

Small - 1000-5000 lines. You know the details of individual modules and functions and within a line range of where to locate a particular set of functionality with ease. You still have deep knowledge of the bulk of the code.

Medium - 5000 to 20000 lines. You know the structure of the entire project in your head and know roughly what module a particular function exists in and how far in to scroll to find it. You're only maintaining details of code you're actively working at this point.

Large - 20000 to 50000 lines. Detail knowledge is limited to immediate work and everything else is having a birds-eye sense of where stuff is in the code. You're working off high-level structural knowledge and will start using tools to pinpoint things you've forgotten the exact location for. This tends to also be the mental threshold of where an experienced programmer can hold the picture of the entire codebase in their head.

BTW; how many lines should be for testing or code comments ?

[Matt Pickering, 30+ yrs prof dev]

Software system may be described by its size (not very accurate), 2 / 2

Very Large - 50000 to 250000 lines. You're at multiple team members at this point and you're treating the codebase as a set of interlocked projects. Coordination for changes is absolutely essential at this point. Only a handful of developers could maintain a complete mental picture of the codebase at this point and would be deep specialists in it having worked on it for years.

Huge - 250000 to 1M lines. Substantial large scale software engineering efforts. Projects are now an interlocked set of loosely or tightly coupled integration points across multiple applications. No individual can generally hold the detail of the project beyond the block diagram level. Blocks on the diagram are very large software efforts in their own right. Too large for anyone to hold in their head as whole at any level of detail.

Massive - Anything north of 1M lines. These are multi-year, engineering discipline level efforts where documentation needed to coordinate and manage change outstrips the software development effort. Operating systems, large scale enterprise products, specialty software code bases that handle mission critical, large scale functionality or high volumes and so on. These code bases tend to have grown to this size and have evolved over years of work. Impossible for anyone to know fully.

[Matt Pickering, 30+ yrs prof dev]

Example: VFS for Git (ex: GVFS)

This project was formerly known as GVFS (Git Virtual File System). It is undergoing a rename to VFS for Git.

VFS stands for Virtual File System. VFS for Git virtualizes the file system beneath your git repo so that git and all tools see what appears to be a normal repo, but VFS for Git only downloads objects as they are needed. VFS for Git also manages the files that git will consider, to ensure that git operations like status, checkout, etc., can be as quick as possible because they will only consider the files that the user has accessed, not all files in the repo.

The Windows code base is approximately 3.5M files and, when checked in to a Git repo, results in a repo of about 300GB. Further, the Windows team is about 4,000 engineers and the engineering system produces 1,760 daily “lab builds” across 440 branches in addition to thousands of pull request validation builds.

https://devblogs.microsoft.com/bharry/the-largest-git-repo-on-the-planet/

https://github.com/microsoft/VFSForGit

Tampereen yliopisto
Tampere University

Example, about Windows repositories

Small repos: Even though GVFS is a product meant for the world's largest repos, GVFS itself has a rather ordinary codebase when it comes to its scale needs. GVFS stats:

Repo size: 10MB packfile, 2MB working directory, 400 files

Team: 10 people

Branches: 300

Build/Test: 30s build, 10s unit tests, 30m functional tests.

Extra large repos: The Windows codebase is in a class of its own. This code includes all of OneCore, Desktop, Server, Xbox, IOT, Mobile, and HoloLens. Along with the fact that this codebase has been around for decades and still supports legacy features all the way back to the beginning of Windows, you can imagine that the codebase is quite large. Windows stats:

Repo size: 100GB packfile, 300GB working directory, 3.5M files

Team: 4000 people

Branches: Estimated to reach between 150K to 250K

Build/Test: A few seconds to few minutes incremental build, 12h full build, a few minutes incremental tests, several hours for full validation.

https://docs.microsoft.com/en-us/azure/devops/learn/git/git-at-scale#extra-large-repos

Tampereen yliopisto
Tampere University

”Average amount” of bugs

Folklore: ”industry average” amount is 15..50 bugs in KLOC.

Microsoft applications; 10..20 defects / KLOC during in-house testing, 0,5 defects / KLOC in released code. [2007]

Folklore: ”A C++ coder will be productive after 1,5 years of working.”

Project size (number of lines of code)	Average error density
less than 2K	0 - 25 errors per 1000 lines of code
2K - 16K	0 - 40 errors per 1000 lines of code
16K - 64K	0.5 - 50 errors per 1000 lines of code
64K - 512K	2 - 70 errors per 1000 lines of code
512K and more	4 - 100 errors per 1000 lines of code

https://hownot2code.com/2017/11/08/search-for-bugs-in-code-at-the-early-stage/

Tampereen yliopisto
Tampere University

Some lines-of-code (LOC) counts

It is difficult to get exact answers, but it is estimated that

- Windows XP: 40 MLOC
- Windows Vista: 50 MLOC
- Windows 7: 40 MLOC
- Windows 8: 50..60 MLOC
- Windows 10: 50 MLOC
- Facebook: 62 MLOC
- Google; more than 2 BLOC, 9 M source files, 25000 developers

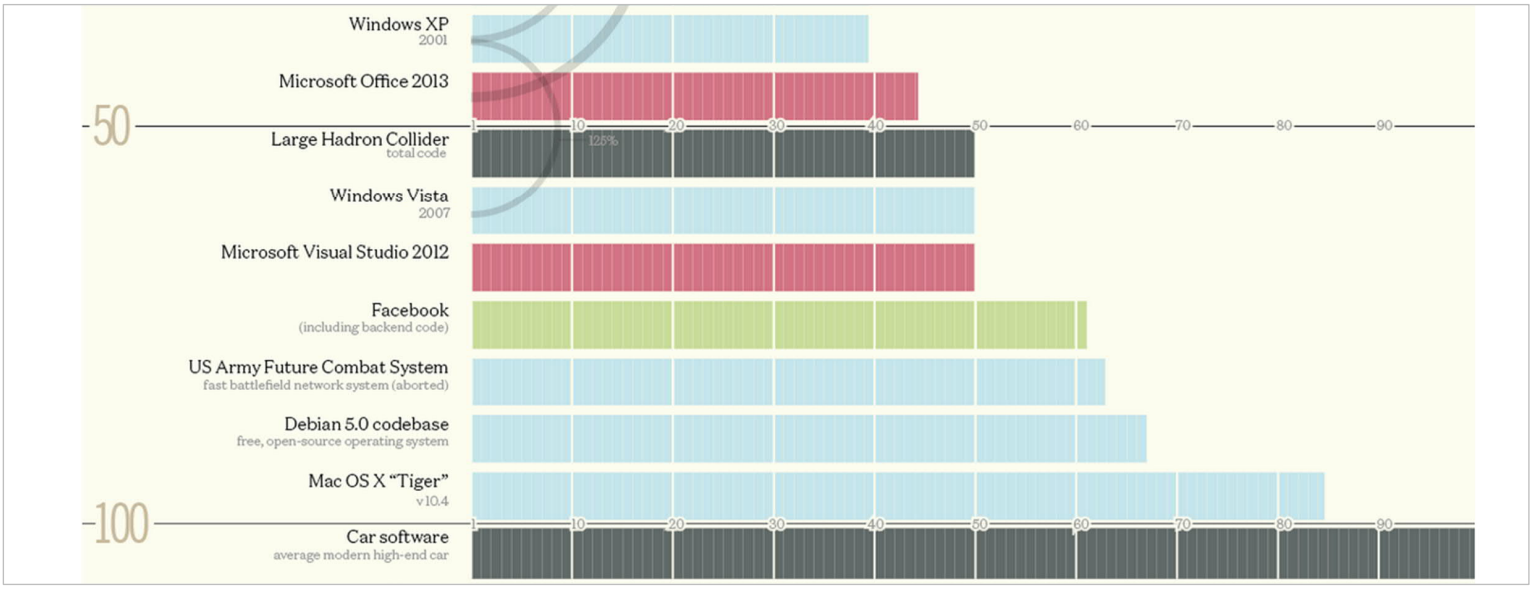
Is only one defect per 1000 lines of code enough for good quality ?

- F-22 fighter; 1,7 MLOC
- Boeing 787; 14 MLOC
- F-35 fighter; 24 MLOC.

Folklore: ”average programmer writes 50 lines of production code in a day”.

Tampereen yliopisto
Tampere University

Some lines of code (LOC) amounts



The later you find bugs, the more costly it is.

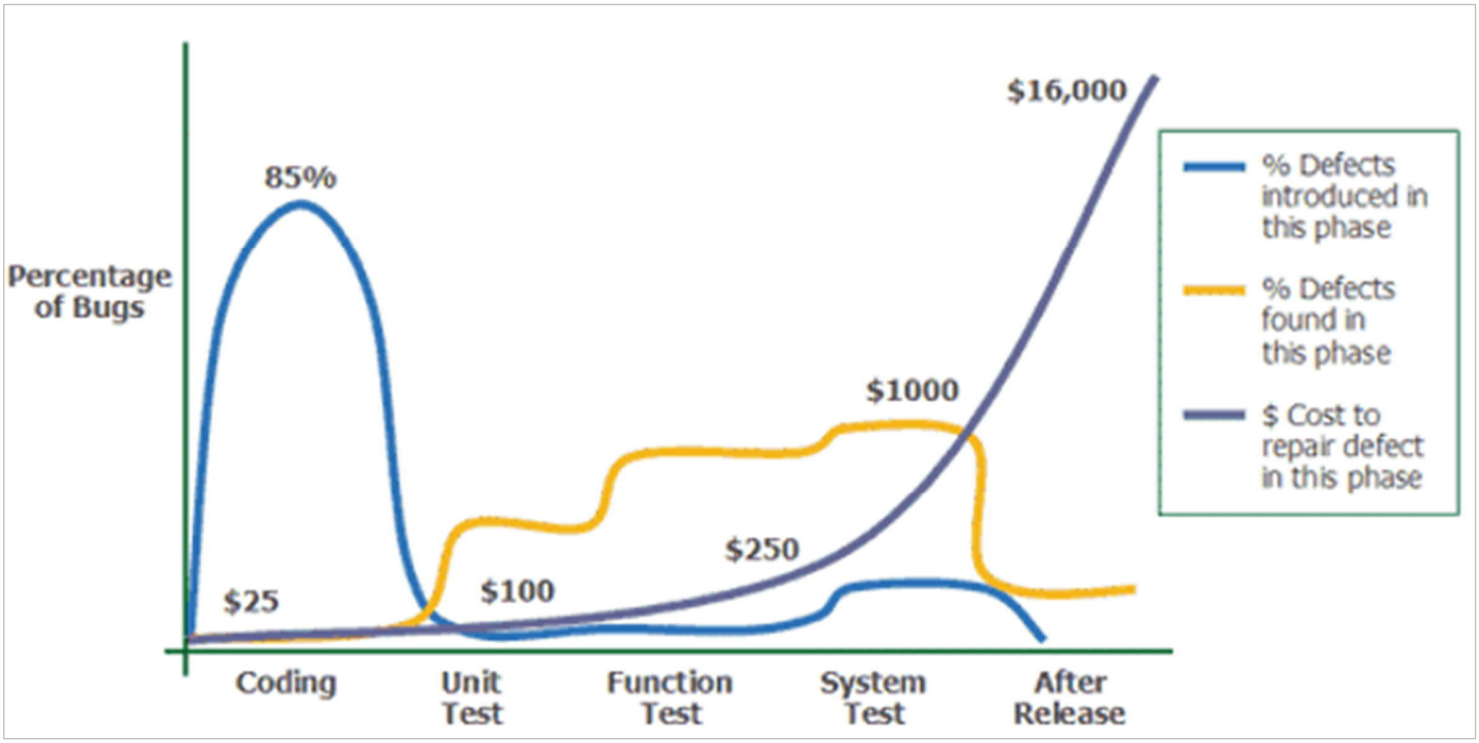


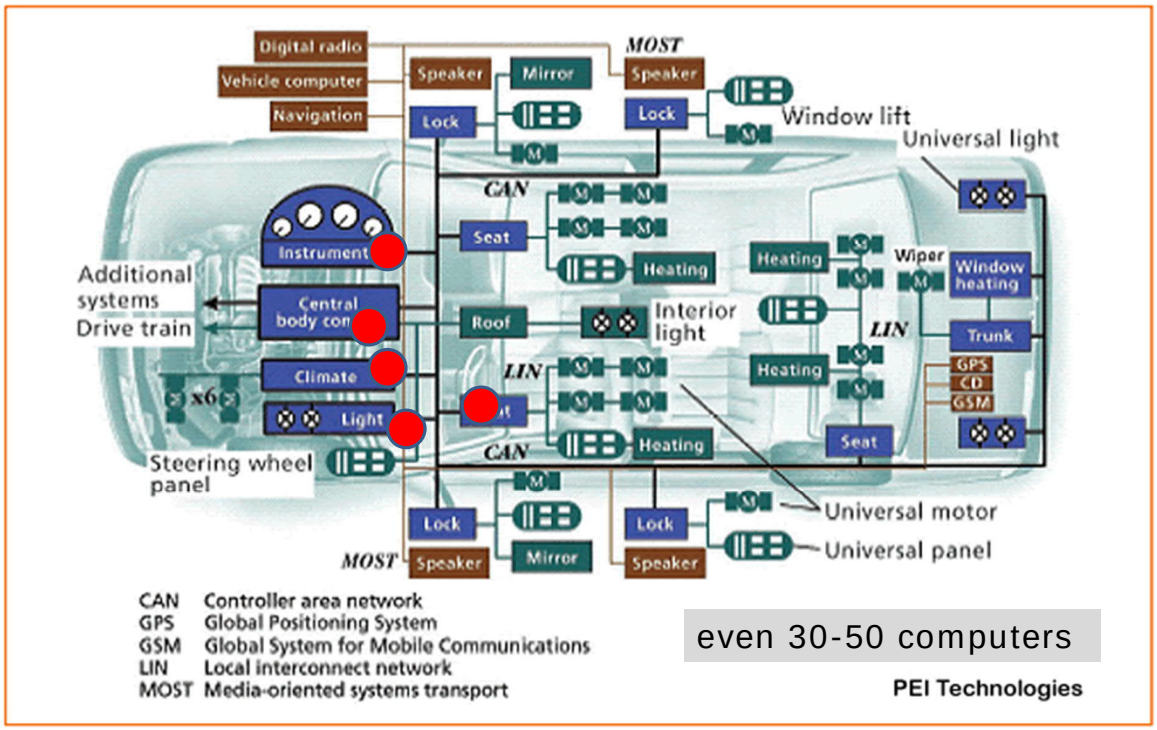
Table 2-1. Allocation of Effort

	Requirements Analysis	Preliminary Design	Detailed Design	Coding and Unit Testing	Integration and Test	System Test
1960s – 1970s	10%			80%	10%	
1980s	20%		60%		20%	
1990s	40%	30%		30%		

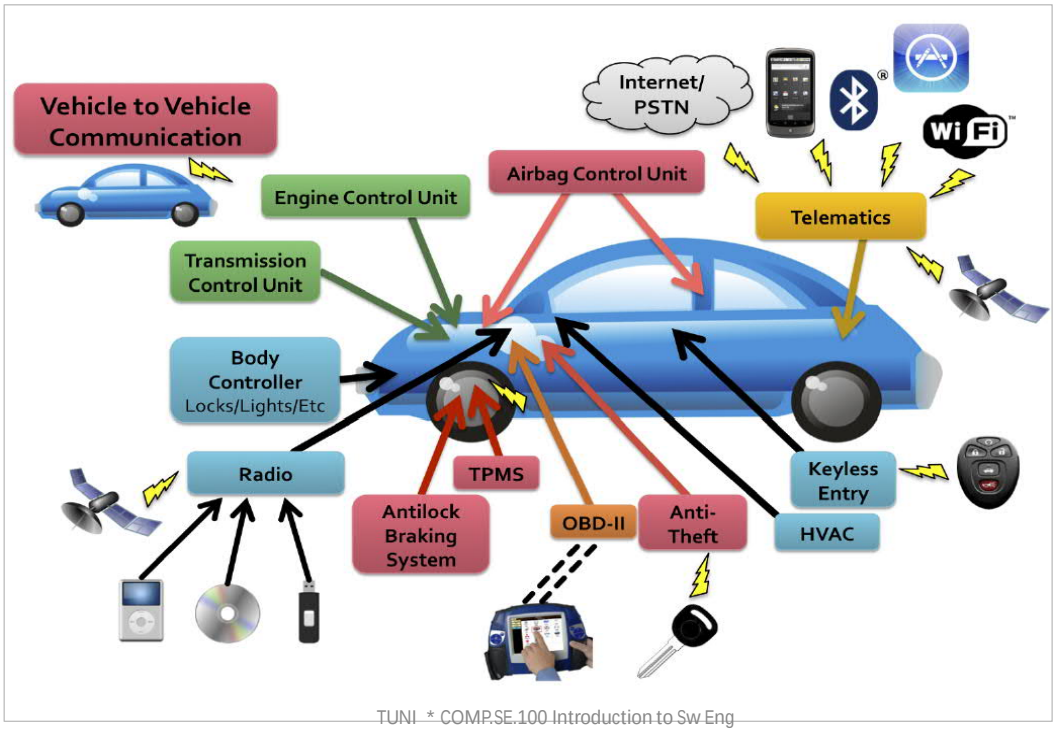
Source: Andersson, M., and J. Bergstrand. 1995. "Formalizing Use Cases with Message Sequence Charts." Unpublished Master's thesis. Lund Institute of Technology, Lund, Sweden.

What is this... a game ?

http://www.aa1car.com/library/can_systems.htm

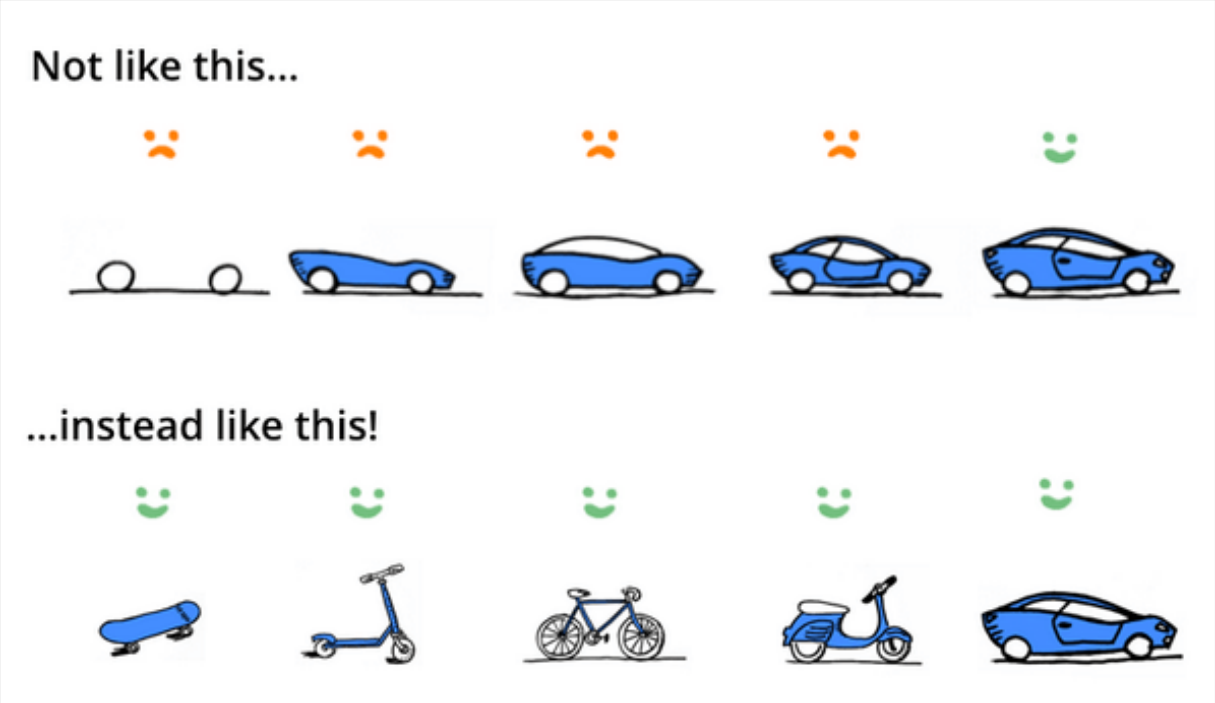


http://www.dailytech.com/Charlie+Miller+Releases+Open+Source+Car+Sabotage+Toolkit/article33308.htm



Tampereen yliopisto
Tampere University

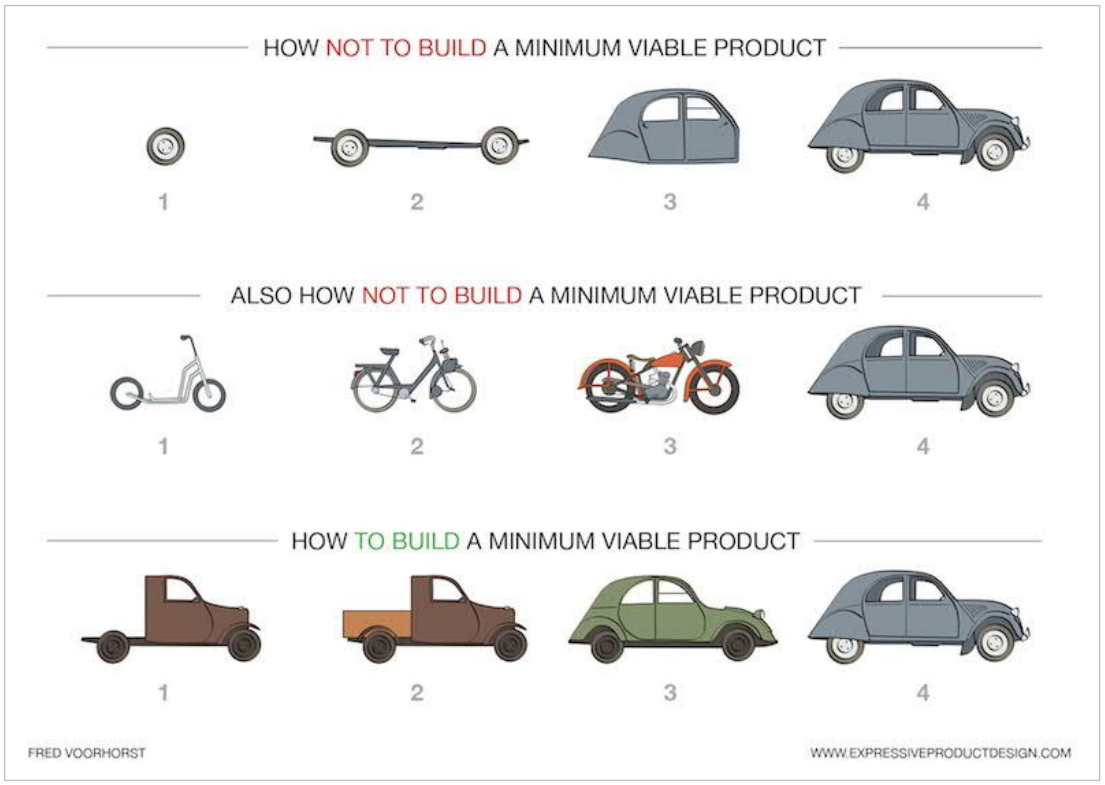
Iterative approach



Tampereen yliopisto
Tampere University

Minumum Viable Product (MVP) is NOT the "quickest hack" you dare to send to customer. This is a common misunderstanding.

MVP is the quickest made product version which brings profit/value to vendor and customer.



Folklore: sw developer's working hour "cost" inside a company may be 70..100 euros.

So... a one-person project lasting six months... equals half a person-year. 1000 working hours yearly ? That makes 500 h x 70 e = 35000 euros.

10 person project, six months, 350000 euros.

Project is much more than just coding...

Plus 25 % profit to vendor ?

And then somebody wonders why software is expensive ??

Senior consultant may take 500..1000 euros / day.

COMPARISON OF SOFTWARE DEVELOPMENT COMPANIES

	Enterprise	Mid-size company	Small team	Freelancers
Size (number of people)	Over 1000	50-1000	Below 50	Individuals
Technologies and skillsets available	Any	Any	Limited	Very limited
On-demand availability	You can hire any specialist (designer, QA, marketer) on a part-time basis	You can hire any specialist (designer, QA, marketer) on a part-time basis	If a specific skill is not available in the team, you have to find a new team	You have to recruit each new specialist individually
Scale up / scale down option	Can easily meet your growing requirements or cut team to the size you need	Can easily meet your growing requirements or cut team to the size you need	Have insufficient resources to keep up with your growth	Never ending process of finding and hiring new freelancers
Administrative issues and overheads (maintenance, sick leaves, hardware requirements)	Covered	Covered	Partly / Uncovered	Uncovered
Control over the development process	Developed project management practices, transparent processes, reports and monitoring	Developed project management practices, transparent processes, reports and monitoring	Very much depends on the team and management	None
Working process organization	Established, conservative and can't be changed	Flexible and can be adjusted for your needs and terms	Poorly organized	None
Customized approach	None	Provided	Provided	Provided
Risks	Low and covered	Low and covered	Medium and uncovered	Extremely high - russian roulette

www.easternpeak.com

agiles

SUPPORT DOWNLOADS

YOUR ROLE SOLUTIONS TECHNOLOGY ABO

24. August 2018

Why ERP projects fail – Lidl stops million-dollar SAP project

ERP systems can help companies to transparently manage process flows in all areas of the organization, better manage resources, and increase the long-term viability of the business. Particularly in large organizations with complex structures and workflows, ERP solutions have become indispensable today. However, from time to time the media report on failed major projects with losses in the millions – as in the case of the cooperation between the discounter Lidl and SAP AG, which raises many questions. Reason enough for us to shed light on the topic.

Consultancy.org Americas Europe Middle East Africa Asia Oceania

Consultancy.uk United Kingdom | Country ▾

Join the platform

News • Consulting Firms • Projects • Events • Jobs • Career • Consulting Industry • Partners

Industries • Services • Research

Filter

Firm

Consulting industry

Functional area

Lidl cancels SAP introduction having sunk €500 million into it

13 August 2018 | Consultancy.uk

Disruptive discount grocery brand Lidl has made an uncharacteristic misfire, shelving a seven year project to introduce SAP to its business. The attempts wasted an estimated €500 million, with Lidl now looking to revive its old system.

Latest news

Non-gaming firms overpay for e-sports brand sponsorship

Charlotte Gregson on Comatch's growth in the UK

Tampereen yliopisto
Tampere University

LIDL project problem;

- want to tailor ERP system to own process
- tailoring is expensive
- tailoring is difficult, PERHAPS company would have better to change work process to fit the (well working) software ??

Read more e.g.:

http://www.bestpracticegroup.com/lidl-cancels-e500m-sap-it-project-4-learnings-to-consider/

Tampereen yliopisto
Tampere University

Software engineering is abstract, 1

Compared to ”traditional” engineering, building software is not concrete.

You can not ”see” it to estimate if it is ready and complete.

You can have some idea about the readiness only by testing, and testing a lot.

It may look good (GUI), but it may have some functionalities not implemented at all.

Two weeks before deadline, you may find one glitch/bug in user interface. The fixing may take three weeks, project gets late two weeks.

To add one simple feature to a product may take two weeks (or even more).

At halfway in a six-month sw project, you may figure out that architecture (basic structure) will not work with planned implementation, so you have to build the software again from scratch.

BTW, if you change tools or libraries during a project... you are unlucky, oh boy.

Software engineering is abstract, 2

In a house building project, you can concretely see how the walls and floors rise up. By just watching the construction site, you can figure out how the house is being built, and you may estimate how long it may take to finish the house. You may watch inside the window and see if the rooms inside are ready or not.

But in software engineering you can not figure out the readiness state of a program just by looking the GUI demo, or code. Some minor feature might take many person-weeks to complete, on the other hand some major change to GUI may just take an hour, you newer know.

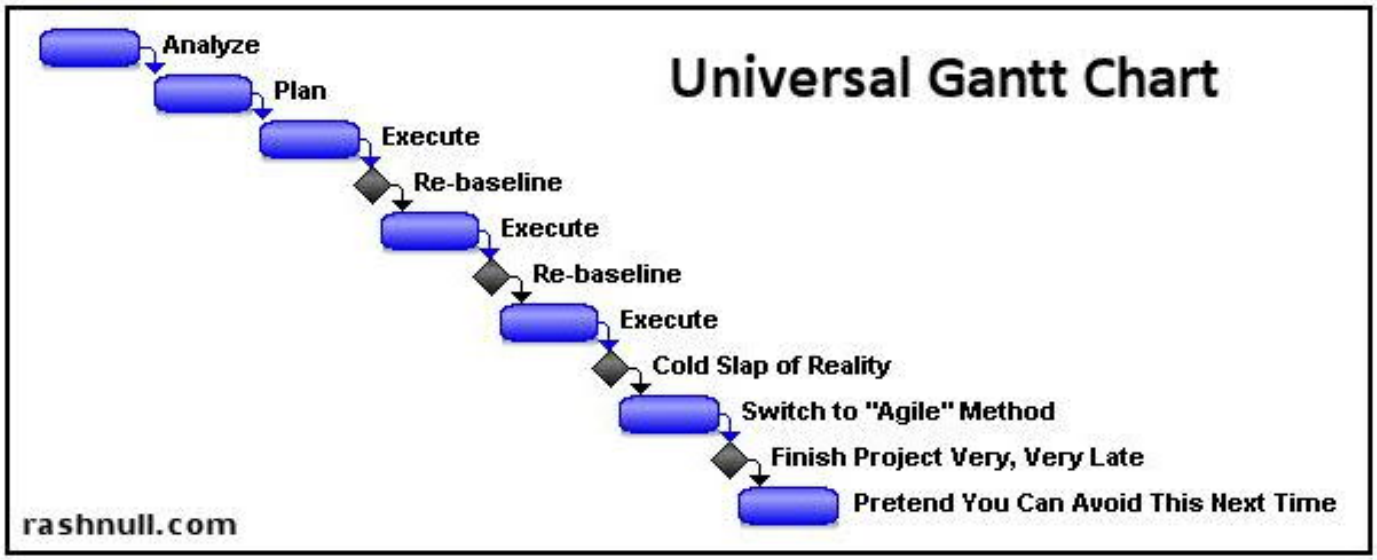
Some small change may be so difficult to make, that it is better to leave it out from the project.

E.g. at car production (line) it is easy to coordinate work, and results are visible

- press shop
- weld shop
- paint shop
- engine shop and TransAxle
- trim/final fitment.

http://www.team-bhp.com/forum/indian-car-scene/83316-automotive-manufacturing-overview.html

Universal Gantt Chart

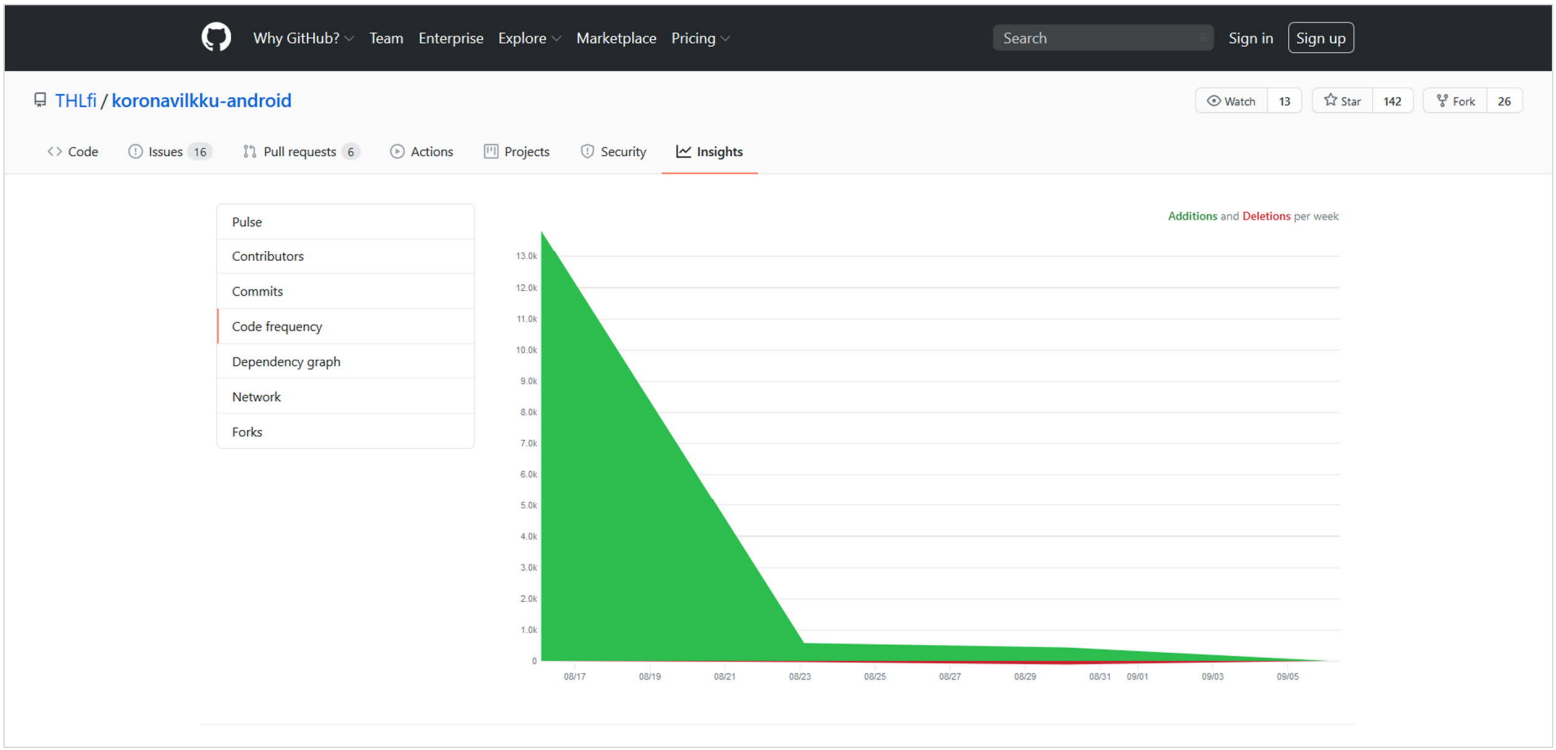


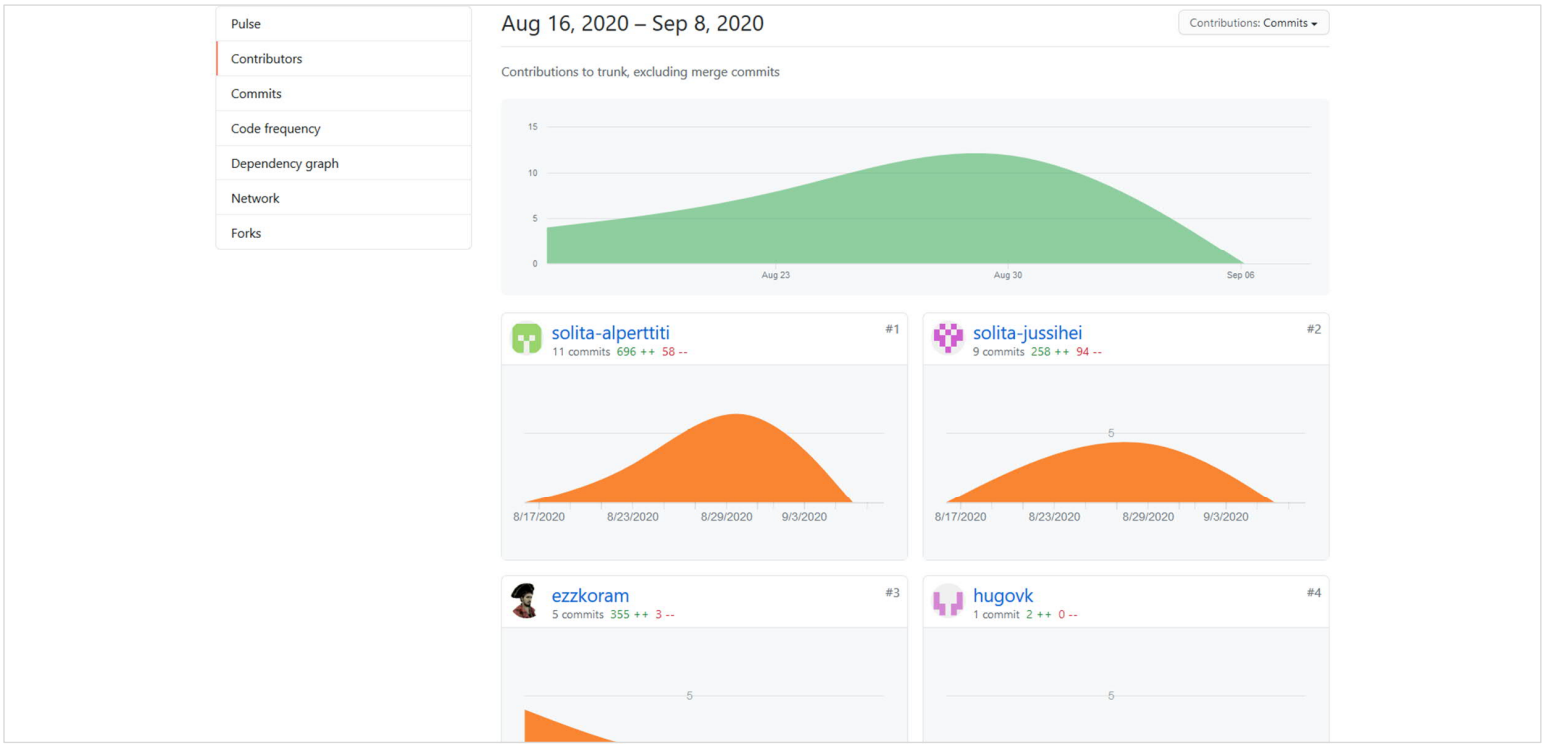
Today's ("hype" ?) software development method(ologie)s

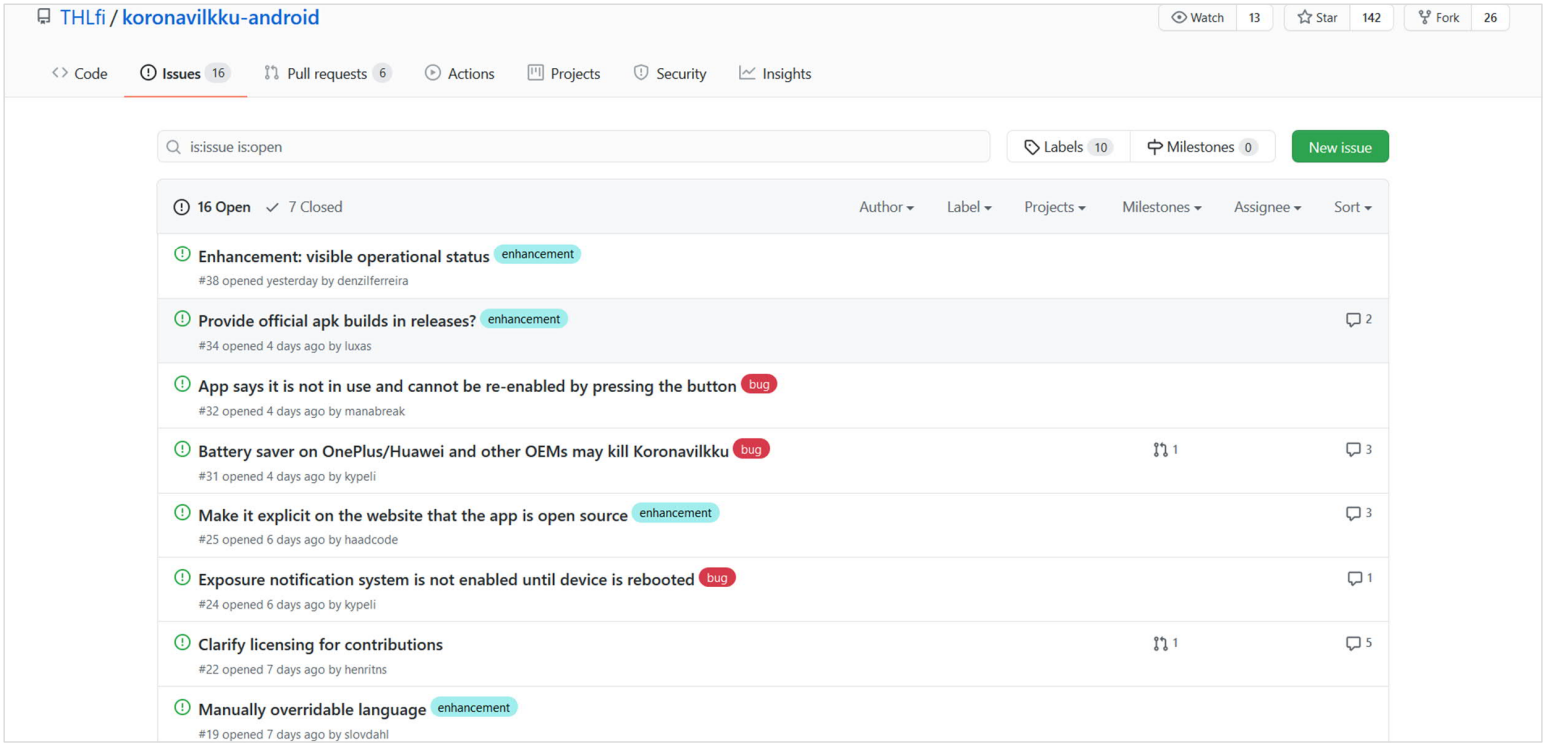
- agile methods (agile, FI: ketterä); iterative
- Scrum (one agile method)
- Kanban; visible board, WIP (pull), optimisation
- Lean = eliminate waste, optimise
- DevOps = DevelopmentOperations (CI, CD)
- SEMAT = Software Engineering Method and Theory
- SAFe = Scaled Agile Framework, for BIG projects
- Blended Agile = combination of two or more Agile-methods
- Hybrid Agile = Agile + non-Agile-method.

The Best Method is the one, which works best for your project.

Methodology is a set on methods (and practices). Keep your eyes and ears open for methods worth trying in your own work.


Why GitHub?
Team
Enterprise
Explore
Marketplace
Pricing
Search
Sign in
Sign up
THLfi / koronavilkku-android
Watch 13
Star 142
Fork 26
Code
Issues 16
Pull requests 6
Actions
Projects
Security
Insights
Pulse
Contributors
Commits
Code frequency
Dependency graph
Network
Forks
Additions and Deletions per week
13.0k
12.0k
11.0k
10.0k
9.0k
8.0k
7.0k
6.0k
5.0k
4.0k
3.0k
2.0k
1.0k
0
08/17
08/19
08/21
08/23
08/25
08/27
08/29
08/31
09/01
09/03
09/05


Pulse
Contributors
Commits
Code frequency
Dependency graph
Network
Forks
Aug 16, 2020 – Sep 8, 2020
Contributions: Commits
Contributions to trunk, excluding merge commits
15
10
5
0
Aug 23
Aug 30
Sep 06
solita-alperttiti
#1
11 commits 696 ++ 58 --
8/17/2020
8/23/2020
8/29/2020
9/3/2020
solita-jussihei
#2
9 commits 258 ++ 94 --
5
8/17/2020
8/23/2020
8/29/2020
9/3/2020
ezzkoram
#3
5 commits 355 ++ 3 --
5
hugovk
#4
1 commit 2 ++ 0 --
5



Tampereen yliopisto
Tampere University

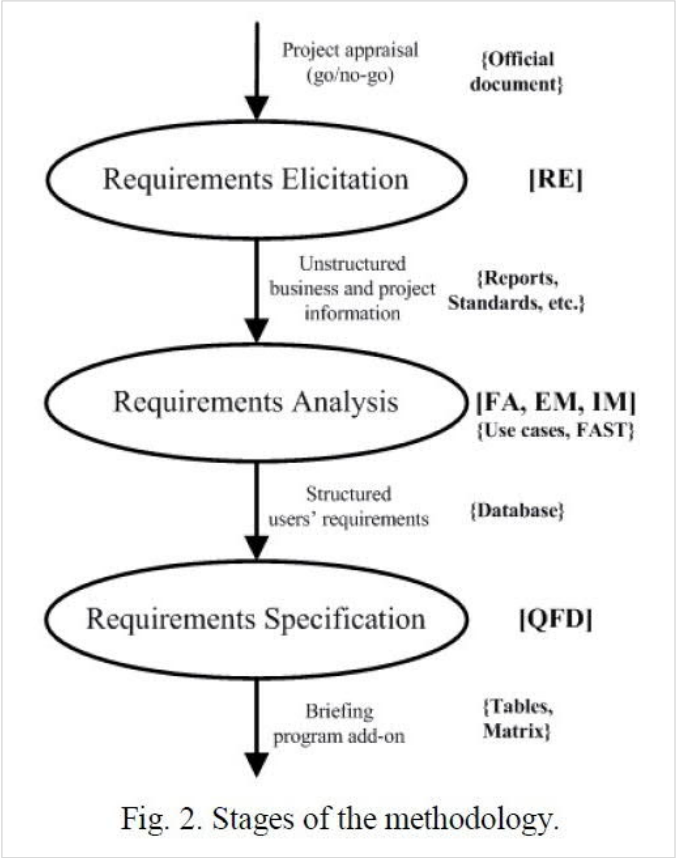


Fig. 2. Stages of the methodology.

RE = Requirements Engineering

FA = Functional Analysis

IM = Information Modelling

FAST = Functional Analysis System Techniques

QFD = Quality Function Deployment

[Improving users satisfaction by using requirements engineering approaches in the conceptual phase of construction projects: the elicitation process, 2010]

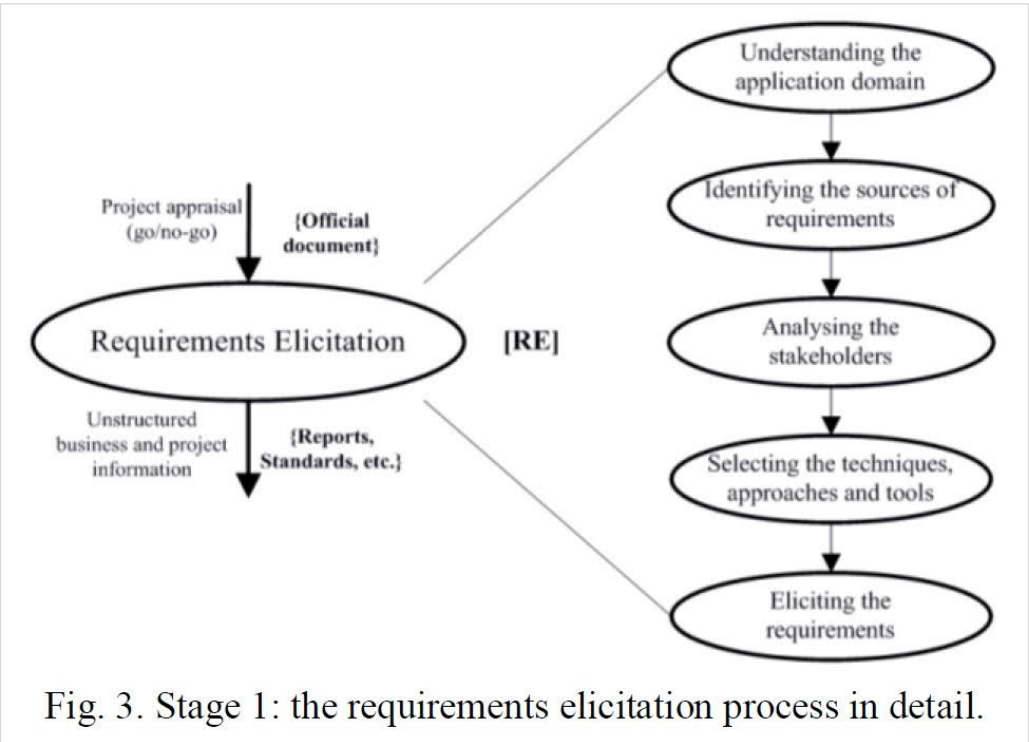


Fig. 3. Stage 1: the requirements elicitation process in detail.

[Improving users satisfaction by using requirements engineering approaches in the conceptual phase of construction projects: the elicitation process, 2010]

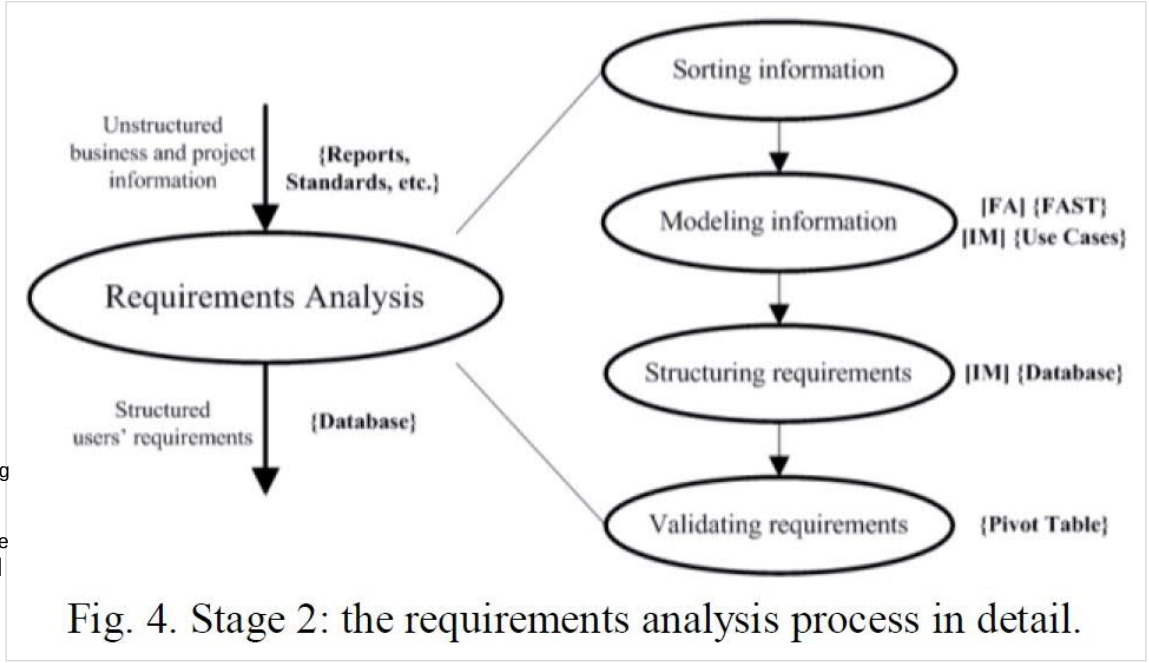


Fig. 4. Stage 2: the requirements analysis process in detail.

[Improving users satisfaction by using requirements engineering approaches in the conceptual phase of construction projects: the elicitation process, 2010]

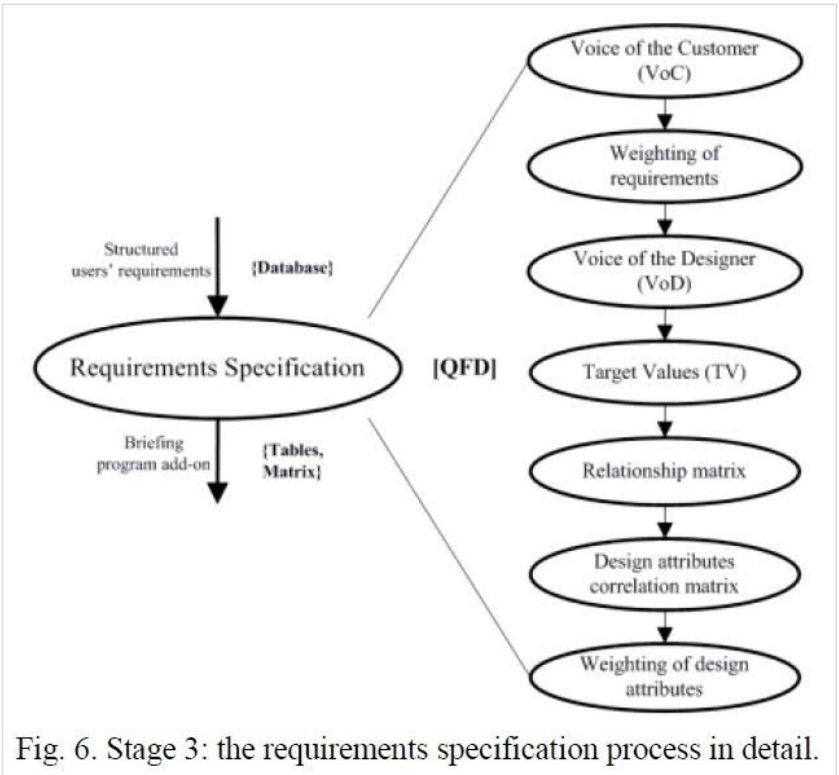


Fig. 6. Stage 3: the requirements specification process in detail.

QFD = Quality Function Deployment

[Improving users satisfaction by using requirements engineering approaches in the conceptual phase of construction projects: the elicitation process, 2010]

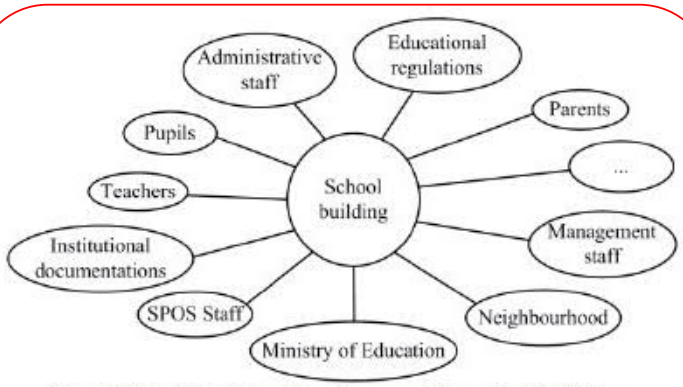


Fig. 6. Part of the interaction diagram of the school building.

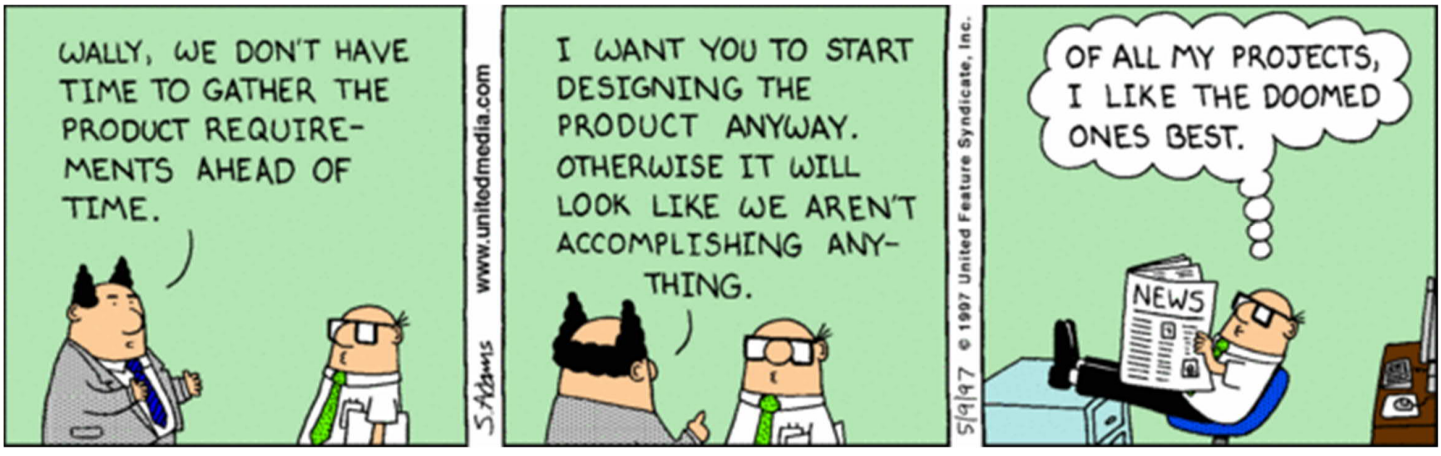
TABLE I
SOURCES WITHHELD AND TYPES

Sources		Type		
Potential	Withheld	Users	Clients	Standards
Pupils		X		
Teachers	X	X		
SPOS staff	X	X		
Administrative staff	X	X		
Management staff	X	X		
Ministry of Education			X	
Educational laws and regulations	X			X
Institutional documentation	X			X

TABLE II
TECHNIQUES WITHHELD FOR ELICITATION PROCESS

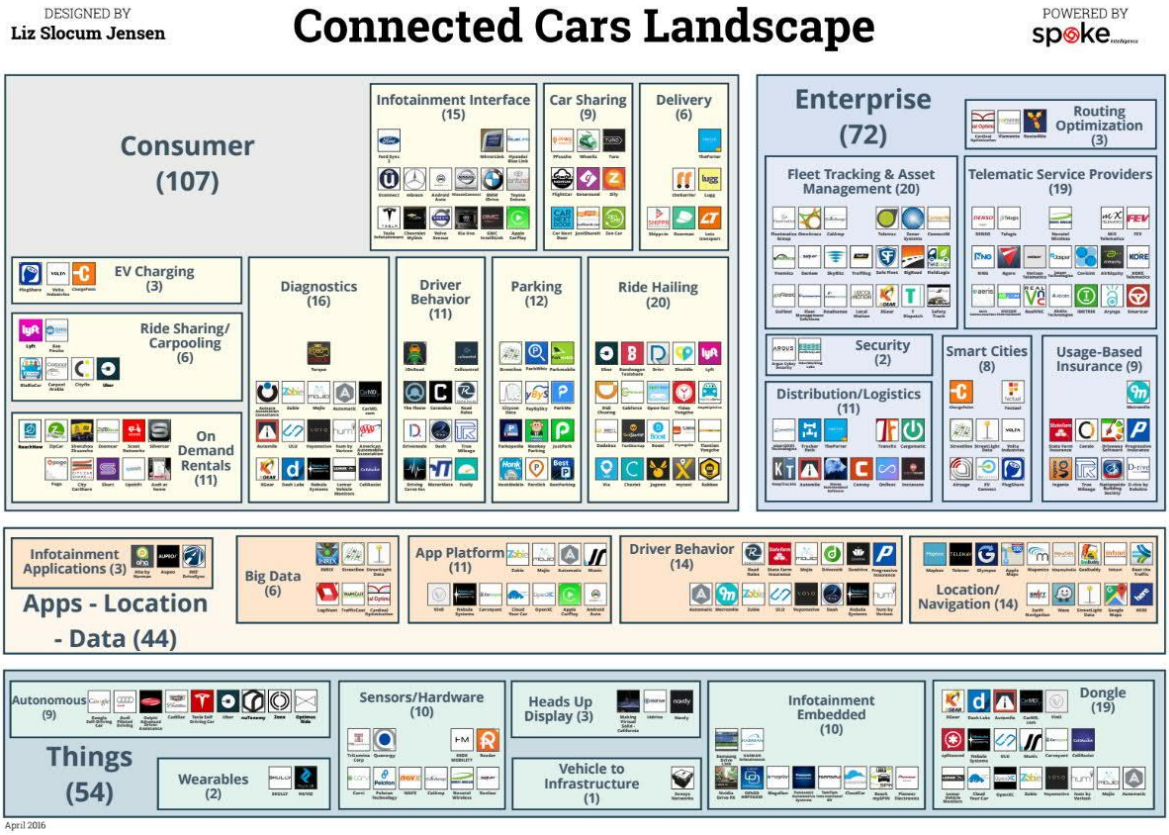
Elicitation process steps	Techniques						
	Domain Analysis	Unstructured Interview	Observation	Brainstorming	Group Work	Task Analysis	Scenarios
Understanding the application domain	X	X	X			X	X
Identifying the sources	X			X	X		
Analyzing the stakeholders		X					
Selecting the techniques	X						
Eliciting the requirements	X	X	X	X	X		

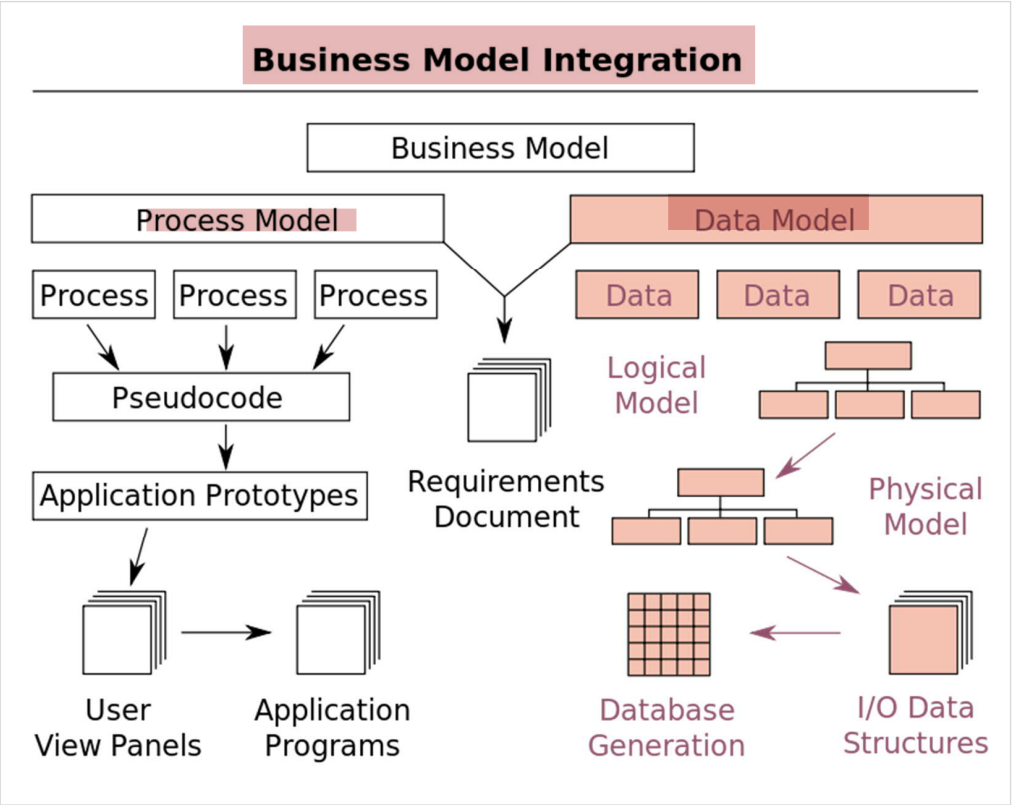
[Improving users satisfaction by using requirements engineering approaches in the conceptual phase of construction projects: the elicitation process, 2010]



In every sw project, there should be "clear and present" business need.
Sw projects should not be started just because competiting company has such, or just because "we can".

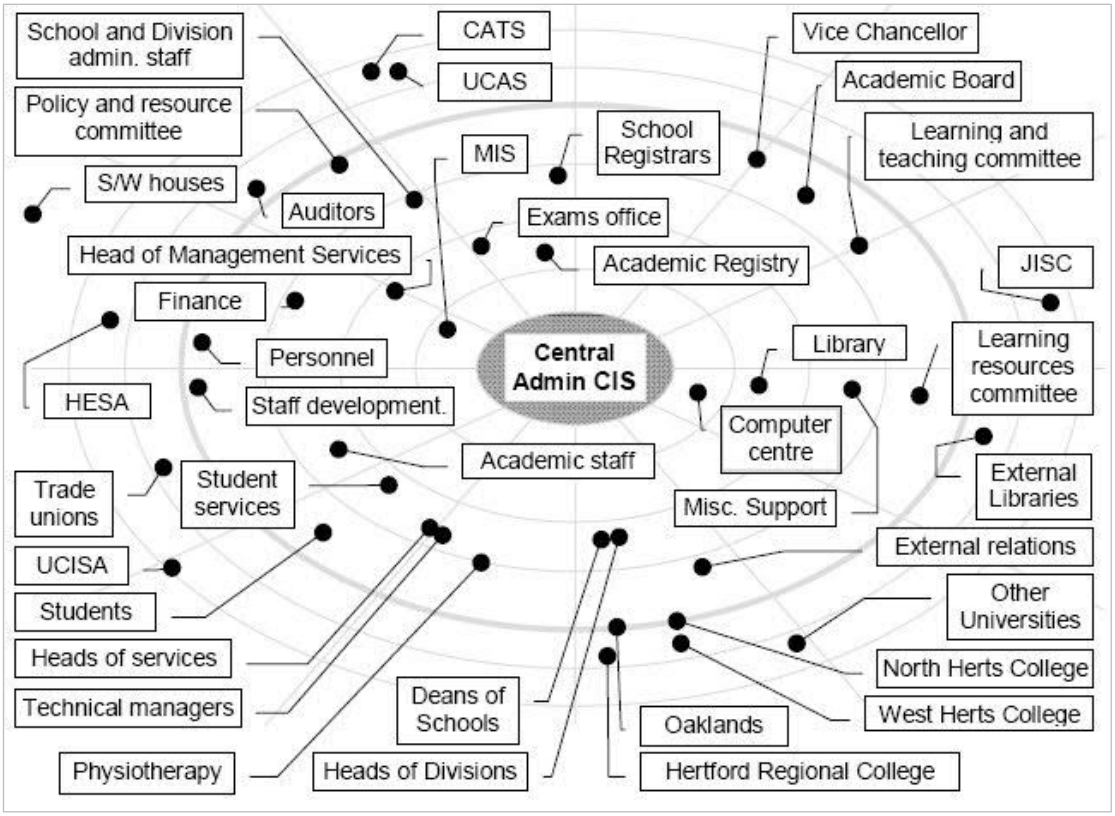
ICT should help companies and people, and to make life better.


DESIGNED BY
Liz Slocum Jensen
Connected Cars Landscape
POWERED BY
spoke
Consumer (107)
Infotainment Interface (15)
Car Sharing (9)
Delivery (6)
EV Charging (3)
Ride Sharing/ Carpooling (6)
On Demand Rentals (11)
Diagnostics (16)
Driver Behavior (11)
Parking (12)
Ride Hailing (20)
Enterprise (72)
Routing Optimization (3)
Fleet Tracking & Asset Management (20)
Telematic Service Providers (19)
Security (2)
Distribution/Logistics (11)
Smart Cities (8)
Usage-Based Insurance (9)
Infotainment Applications (3)
Apps - Location - Data (44)
Big Data (6)
App Platform (11)
Driver Behavior (14)
Location/ Navigation (14)
Autonomous (9)
Things (54)
Wearables (2)
Sensors/Hardware (10)
Heads Up Display (3)
Vehicle to Infrastructure (1)
Infotainment Embedded (10)
Dongle (19)
April 2016



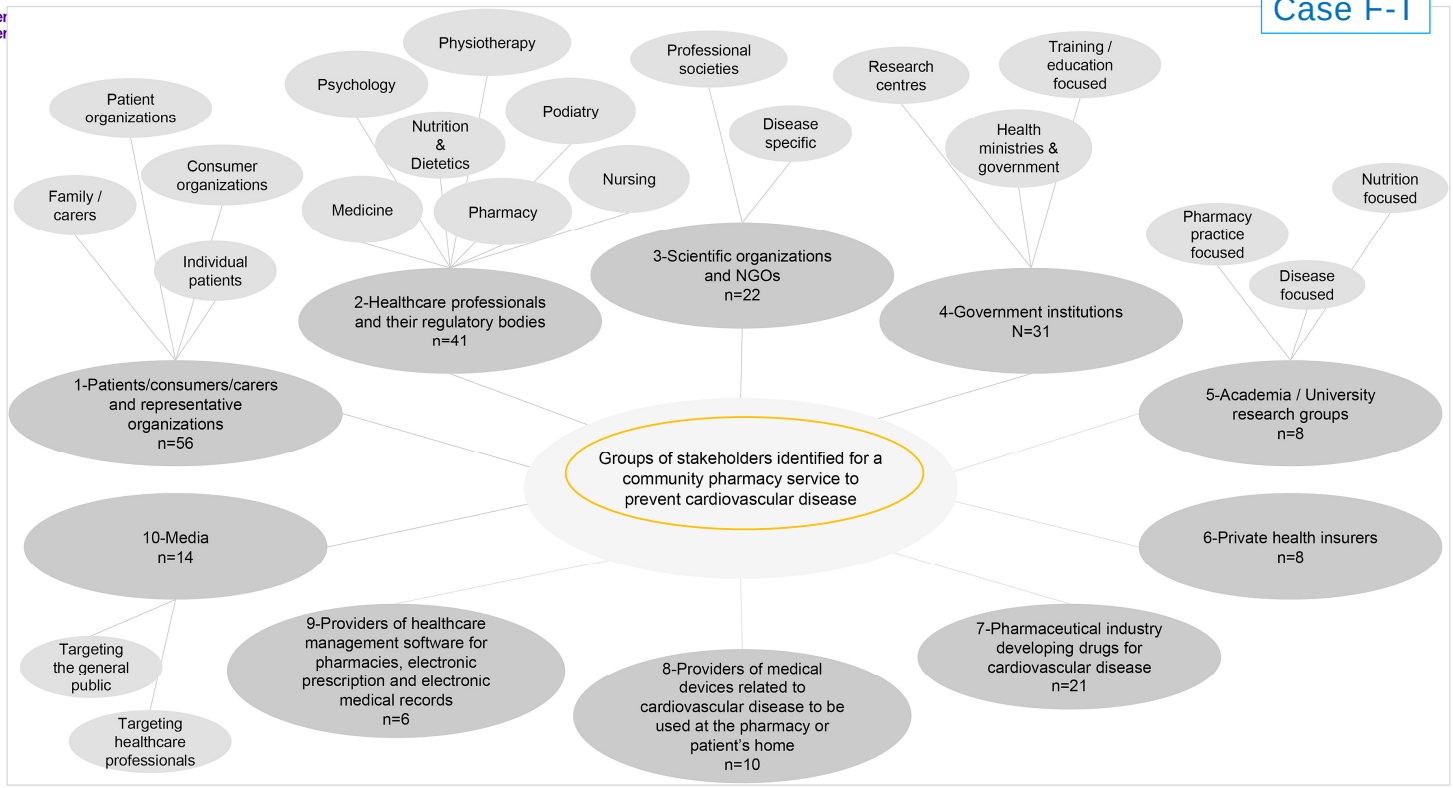
[Paul R. Smith. Redrawn by Marcel Douwe Dekker]

University example



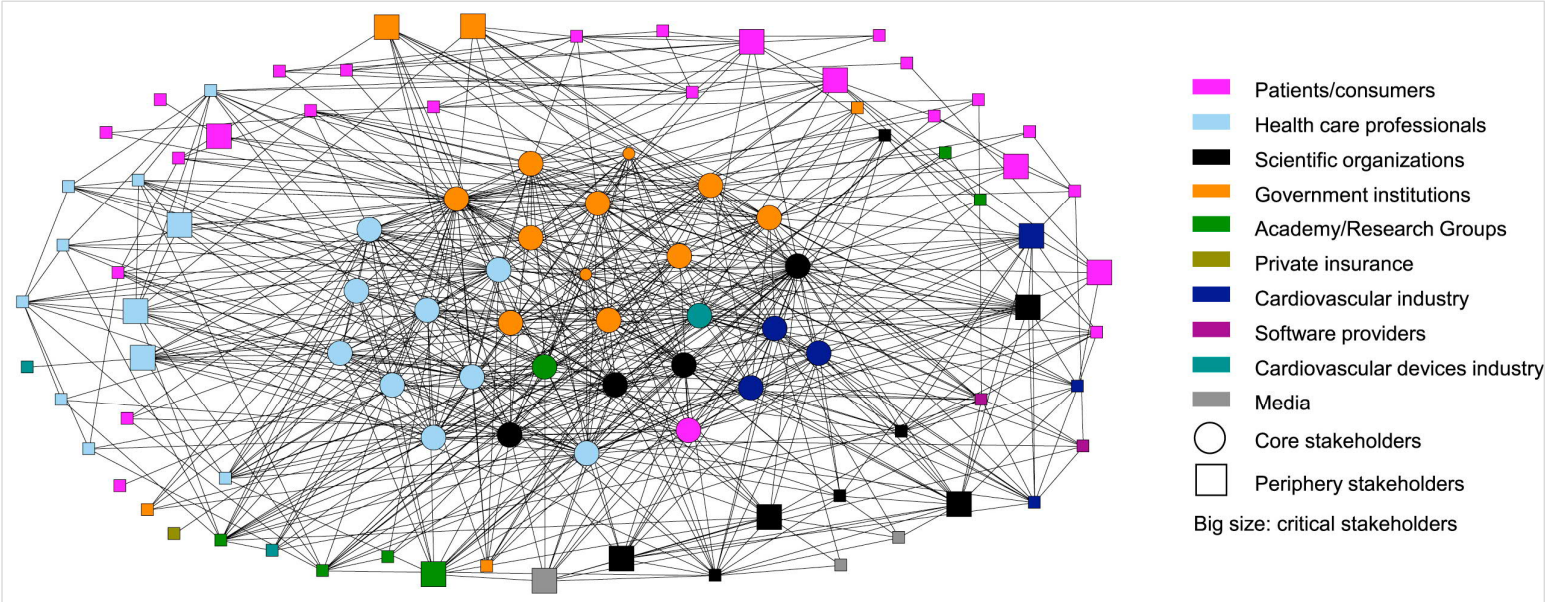
[scenarioplus.org.uk]

Case F-T



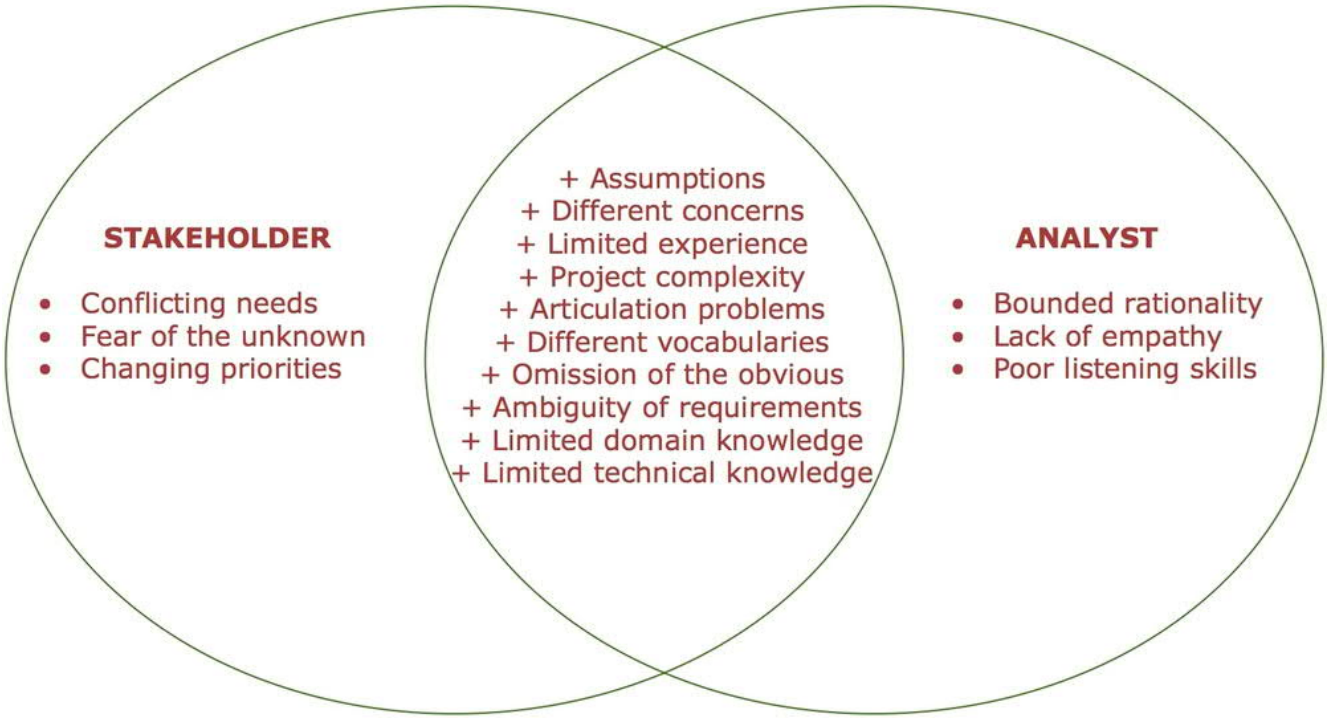
Tampereen yliopisto
Tampere University

Case F-T: Collaboration network about stakeholders.

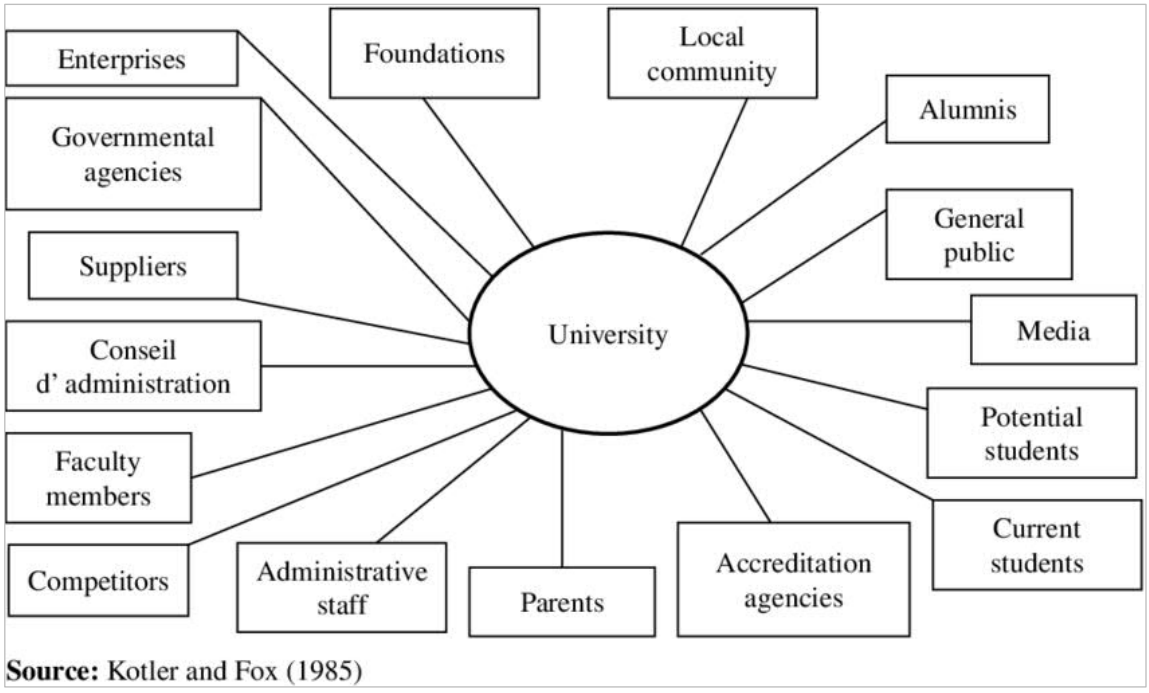


[L. Franco-Trigo et al.: Collaborative health service planning: A stakeholder analysis with social network analysis to develop a community pharmacy service, 2019]

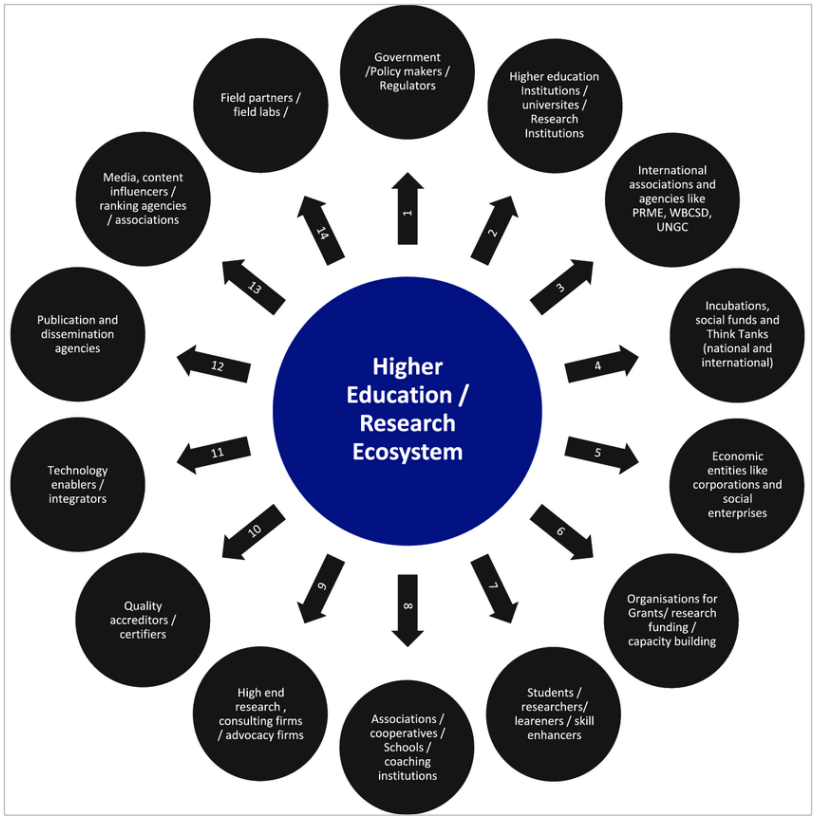
A Snapshot Of The Difficulties Inherent In Requirements Elicitation



[A beginner's guide to requirements elicitation, 2015]



Source: Kotler and Fox (1985)



[Mainstreaming University and Social Enterprise Ecosystem, 2017]

Tampereen yliopisto
Tampere University

Weeding out requirements from stakeholders… remember to be polite.

ISO 24765:2017 Systems and software engineering — Vocabulary

3.1582 feature

- 1. distinguishing characteristic of a system item
- 2. functional or non-functional distinguishing characteristic of a system, often an enhancement to an existing system
- 3. abstract functional characteristic of a system of interest that end-users and other stakeholders can understand

3.1716 functionality

- 1. capabilities of the various computational, user interface, input, output, data management, and other features provided by a product
- Note 1 to entry: This characteristic is concerned with what the software does to fulfill needs. The software quality characteristic functionality can be used to specify or evaluate the suitability, accuracy, interoperability, security, and compliance of a function.

Tampereen yliopisto
Tampere University

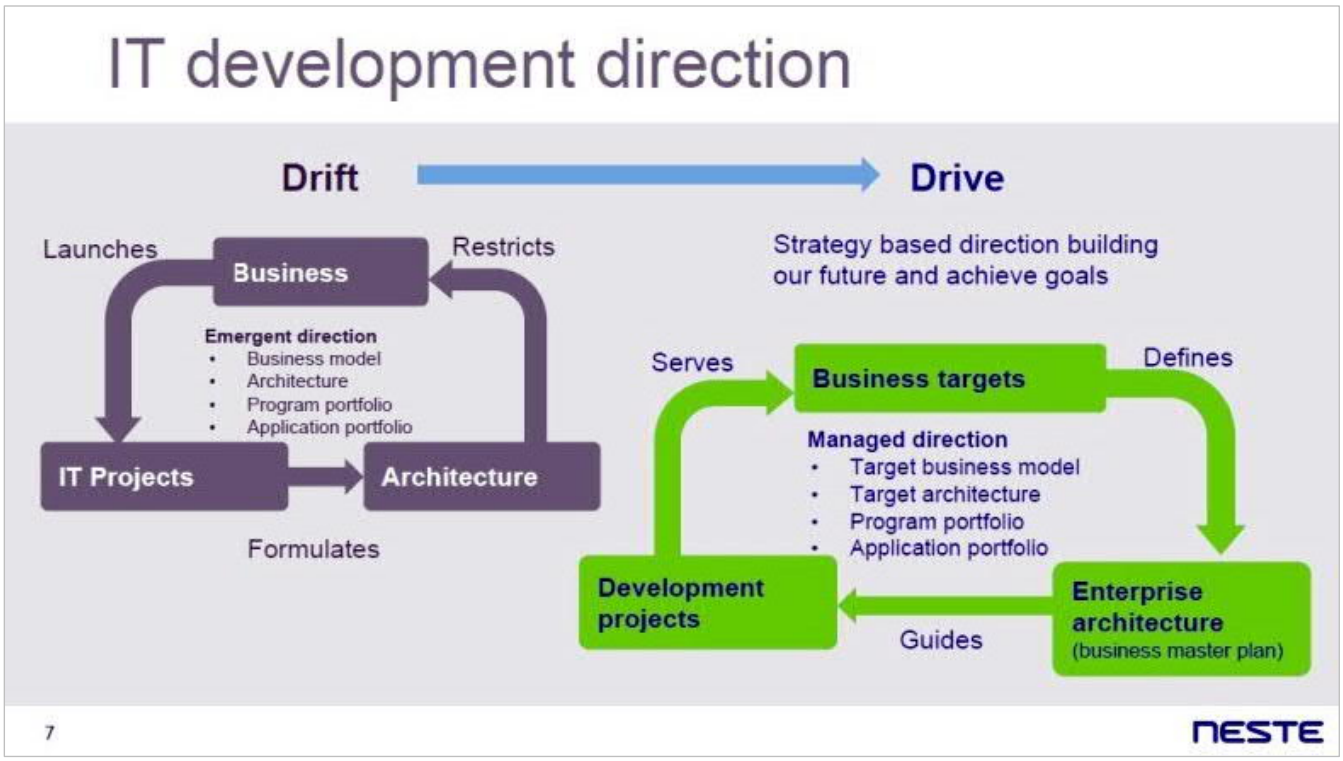
Some other points about sw projects

- Sw projects are not scalabe; what works in five person project, might not work in 50 person project. If one developer/coder produces 100 LOC in one day, 10 developers do not necessarily product 1 KLOC of working and tested code.
- In larger groups/teams, work division and communication takes more and more time. Even at TIE-PROJ groups of seven students complain about difficulty to agree meeting times, to get everybody around the same table (F-2-F).
- programming language should be selected according to project characters, as well as the methods used in the project (one method is not good for all project types)
- sometimes customer suggests/requires some programming language and/or method to be used, such cases should be discussed in more detail (are those wishes based on real facts or experience, or just hype).
- Take end-users into account, ask how they want to do their work, show GUI sketches early. Developers do not know what is best for end-users. Without end-user involvement, some features/functionalities may be forgotten.

”There are only two industries that call their customers ’users’; illegal drugs and software.”

Tampereen yliopisto
Tampere University

ICT projects should be made to help business



[CIO-webinar 21.11.2018]

One essential difference

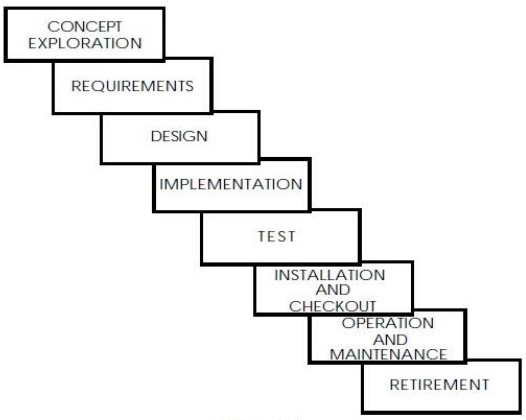


Fig 15
Sample Software Life Cycle

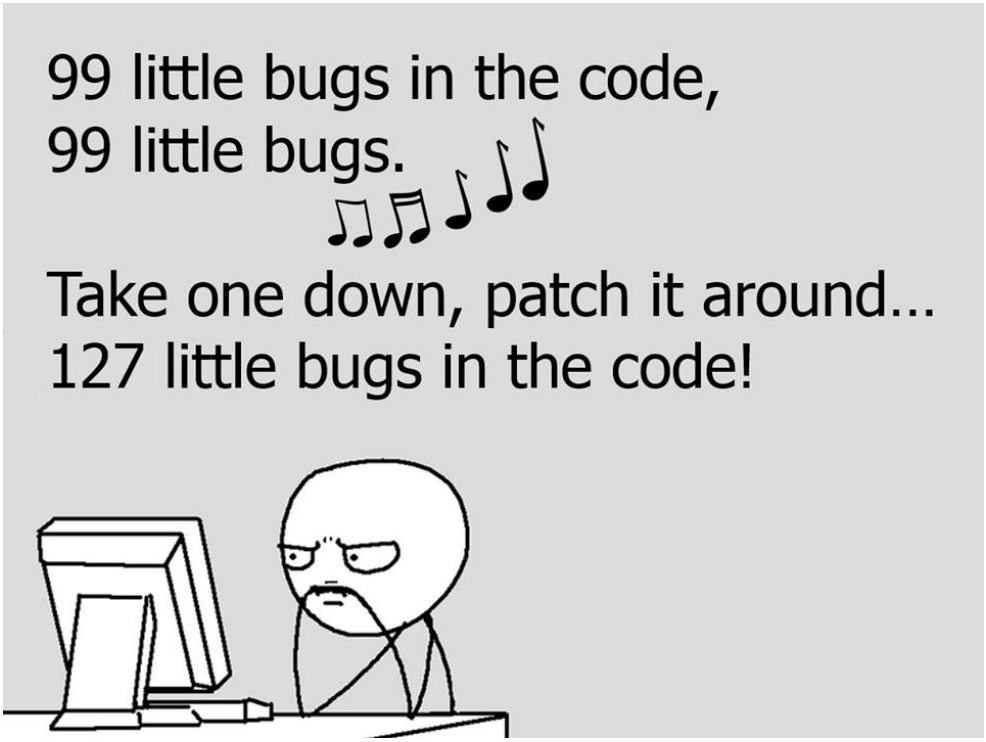
In larger traditional ”waterfall” development process different phases were usually done by different engineers or teams. Such a way tacit information (FI: hiljainen tieto) did not reach next phase; at least slight misunderstandings were common.

While at agile development the same team is doing all the process-related tasks; requirements processing to code and testing.

[IEEE Standard Glossary of Software Engineering Terminology, Std 610.12-1990(R2002)]

Sometimes, especially in old legacy code which has very little if any documentation and very few comment lines, it may happen that fixing one bug causes several other bugs as a side effect.

Maintaining old ”spaghetti” code could be a nightmare. In many cases it would be better to start making a new systems from scratch.



This can only happen in Sw Eng

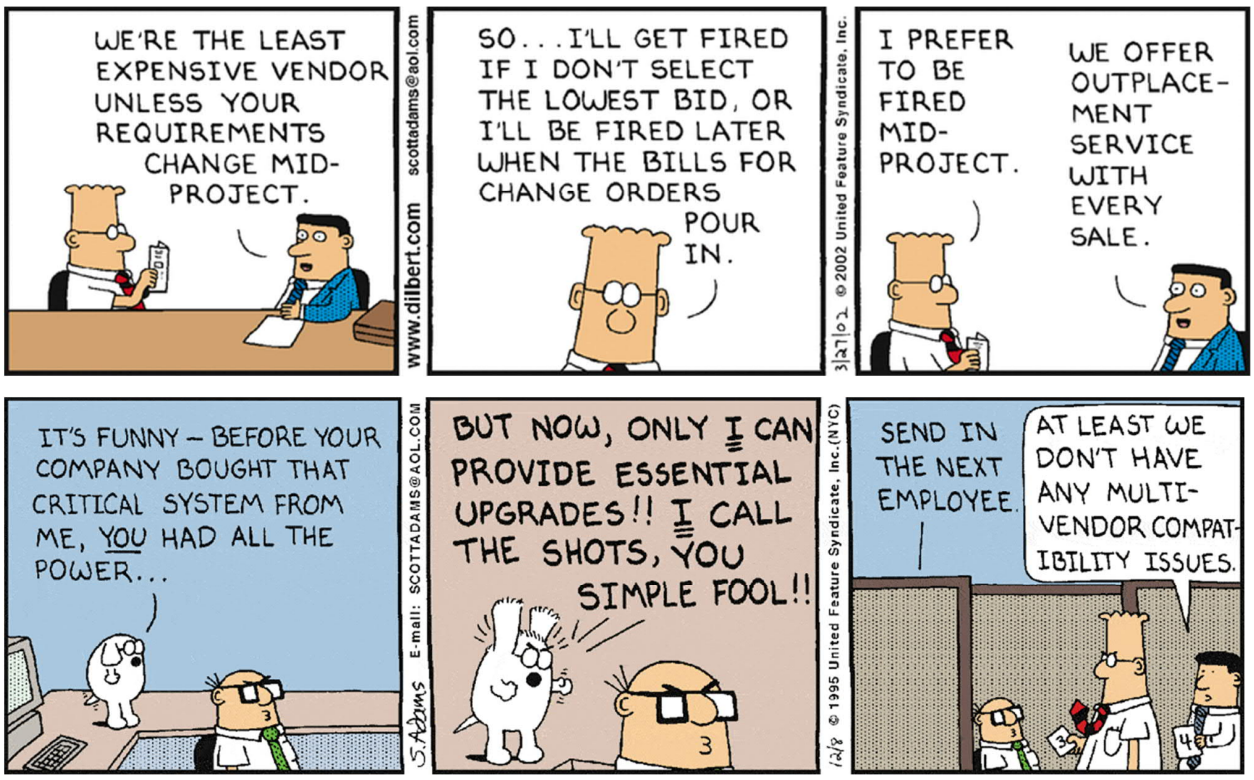
Let there be a 15 week long sw project (estimated).

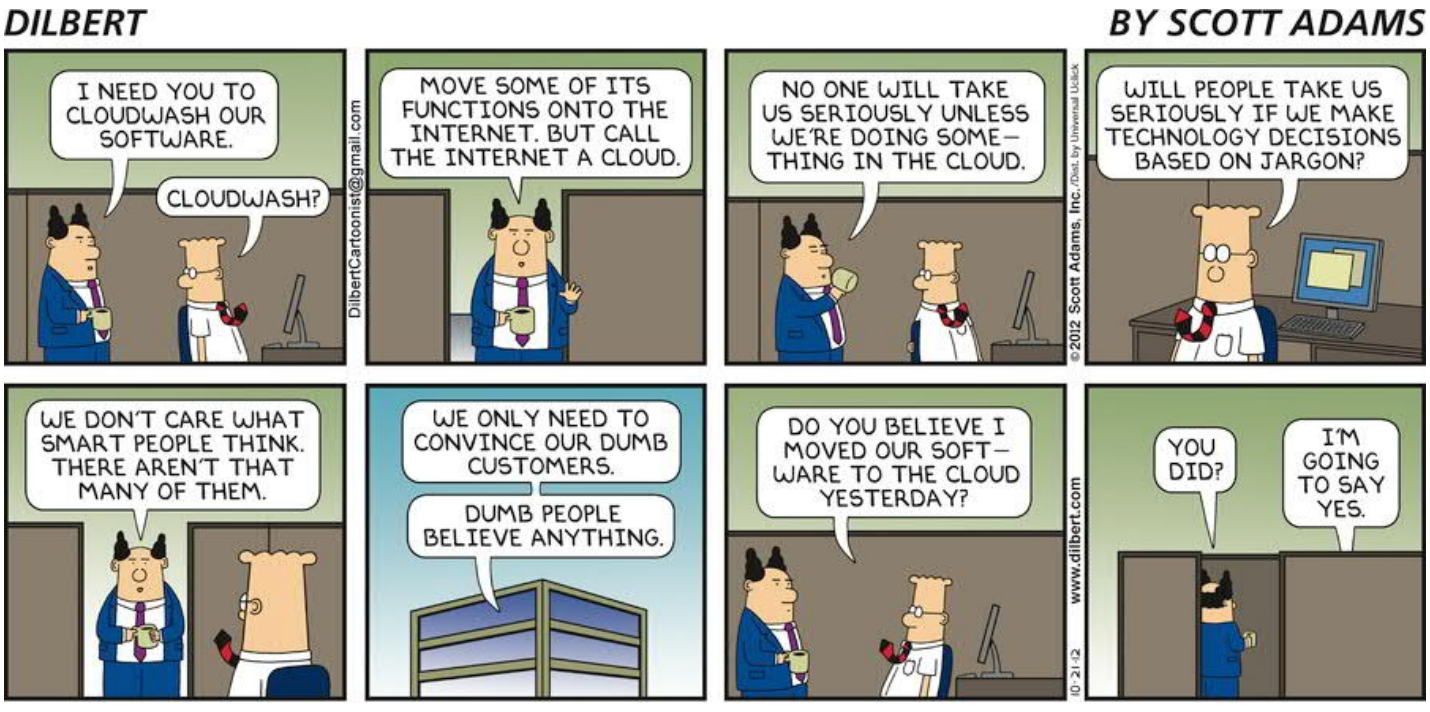
- at week 7 swengineers say it is 40 % ready
- w9; 50 % ready
- w10; 60 %
- w11; 70 %
- w12; 80 %
- w13; 85 %
- w14; 90 %
- w15; 92 %
- w16; 93 %
- w17; 93,5 % ready. Nobody knows what that means ?? Anything working ??

This is a horror story about an old poor sw project.

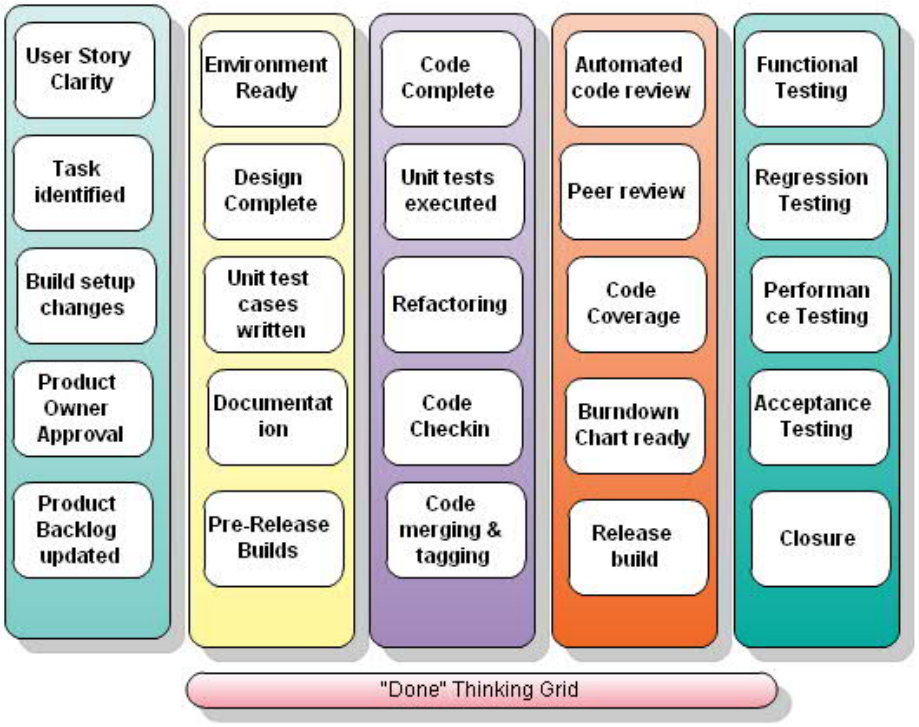
Better than percentage, use features/functionalities to measure readiness.

Single source = ”vendor lock”


WE'RE THE LEAST EXPENSIVE VENDOR UNLESS YOUR REQUIREMENTS CHANGE MID-PROJECT.
www.dilbert.com scottadams@aol.com
SO...I'LL GET FIRED IF I DON'T SELECT THE LOWEST BID, OR I'LL BE FIRED LATER WHEN THE BILLS FOR CHANGE ORDERS POUR IN.
I PREFER TO BE FIRED MID-PROJECT.
WE OFFER OUTPLACE-MENT SERVICE WITH EVERY SALE.
3|27|02 © 2002 United Feature Syndicate, Inc.
IT'S FUNNY – BEFORE YOUR COMPANY BOUGHT THAT CRITICAL SYSTEM FROM ME, YOU HAD ALL THE POWER...
S. Adams E-mail: SCOTTADAMS@AOL.COM
BUT NOW, ONLY I CAN PROVIDE ESSENTIAL UPGRADES!! I CALL THE SHOTS, YOU SIMPLE FOOL!!
12/8 © 1995 United Feature Syndicate, Inc.(NYC)
SEND IN THE NEXT EMPLOYEE.
AT LEAST WE DON'T HAVE ANY MULTI-VENDOR COMPAT-IBILITY ISSUES.

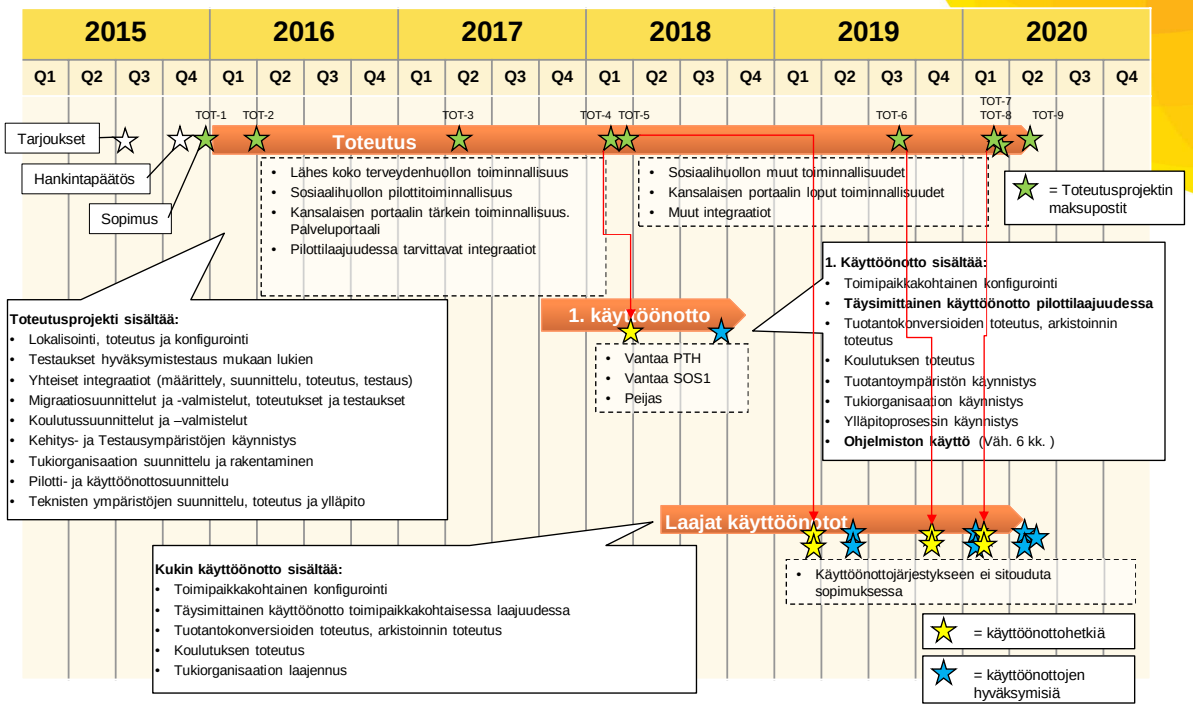

DILBERT
BY SCOTT ADAMS
I NEED YOU TO CLOUDWASH OUR SOFTWARE.
CLOUDWASH?
DilbertCartoonist@gmail.com
MOVE SOME OF ITS FUNCTIONS ONTO THE INTERNET. BUT CALL THE INTERNET A CLOUD.
NO ONE WILL TAKE US SERIOUSLY UNLESS WE'RE DOING SOME-THING IN THE CLOUD.
©2012 Scott Adams, Inc./Dist. by Universal Uclick
WILL PEOPLE TAKE US SERIOUSLY IF WE MAKE TECHNOLOGY DECISIONS BASED ON JARGON?
WE DON'T CARE WHAT SMART PEOPLE THINK. THERE AREN'T THAT MANY OF THEM.
WE ONLY NEED TO CONVINCE OUR DUMB CUSTOMERS.
DUMB PEOPLE BELIEVE ANYTHING.
DO YOU BELIEVE I MOVED OUR SOFT-WARE TO THE CLOUD YESTERDAY?
www.dilbert.com
10-21-12
YOU DID?
I'M GOING TO SAY YES.

A general definition of done (DoD) can be spelled out in a “done thinking grid”



[www.scrumalliance.org]

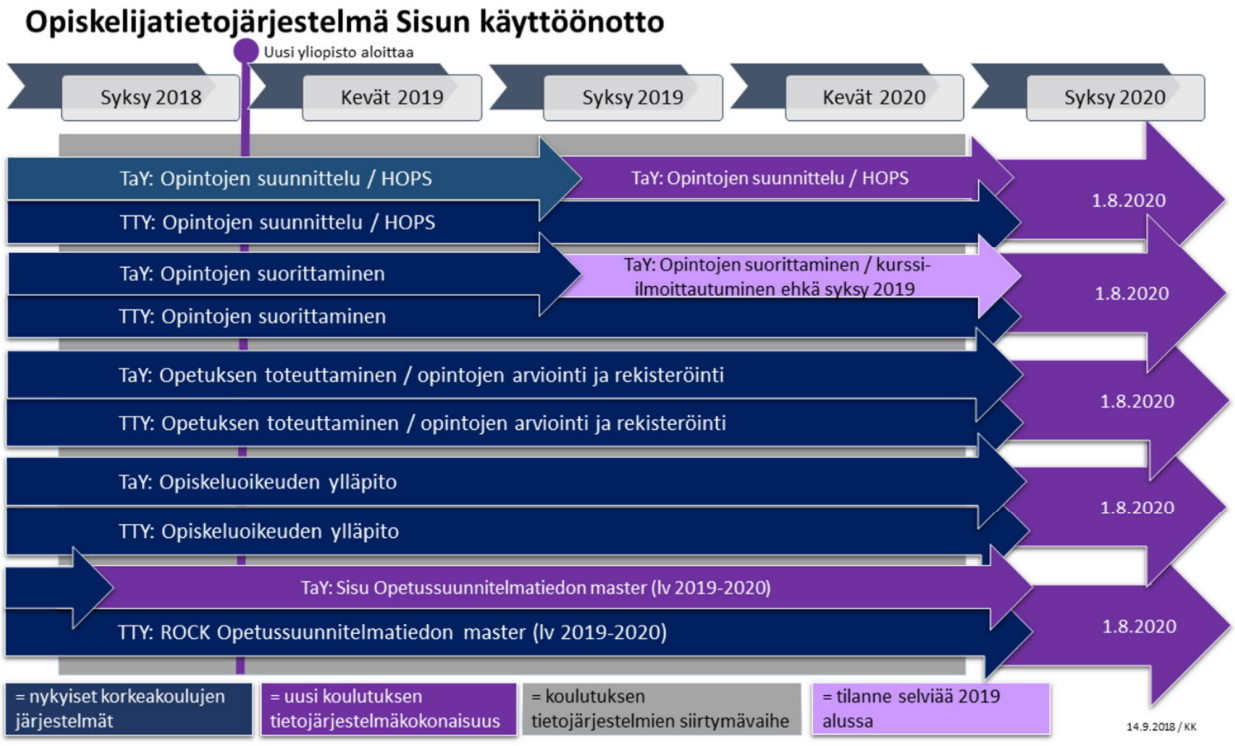
Toteutus- ja käyttöönotto – Pääprojektit ja aikatauluarvio (päivitetty tarjouksen perusteella)



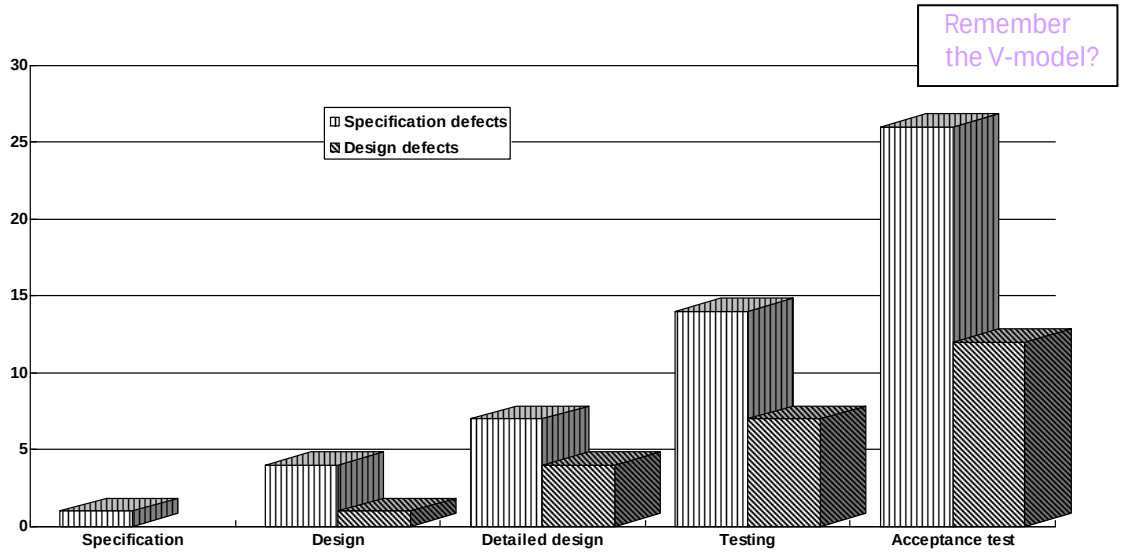
Tampereen yliopisto
Tampere University

https://www.apotti.fi/en/project-follow-up/

Tampereen yliopisto
Tampere University



Relative cost of repairing a defect



[Jalote: an Interactive Approach to Software Engineering, 1991]

Tampereen yliopisto
Tampere University

Tampereen yliopisto
Tampere University

IEEE Code of Ethics [https://www.computer.org/education/code-of-ethics]

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. **PUBLIC** – Software engineers shall act consistently with the public interest.
2. **CLIENT AND EMPLOYER** – Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. **PRODUCT** – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. **JUDGMENT** – Software engineers shall maintain integrity and independence in their professional judgment.
5. **MANAGEMENT** – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. **PROFESSION** – Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. **COLLEAGUES** – Software engineers shall be fair to and supportive of their colleagues.
8. **SELF** – Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Tampereen yliopisto
Tampere University

IEEE Code of Ethics

1. to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, and to disclose promptly factors that might endanger the public or the environment;

2. to avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist;

3. to be honest and realistic in stating claims or estimates based on available data;

4. to reject bribery in all its forms;

5. to improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies, including intelligent systems;

6. to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;

7. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;

8. to treat fairly all persons and to not engage in acts of discrimination based on race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or gender expression;

9. to avoid injuring others, their property, reputation, or employment by false or malicious action;

10. to assist colleagues and co-workers in their professional development and to support them in following this code of ethics.

Tampereen yliopisto
Tampere University

Now the additional L2
extra slides set ends here